

CSS532(Final Project 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, connect the sensor as follows:

VCC to the 5V pin on the Raspberry Pi.

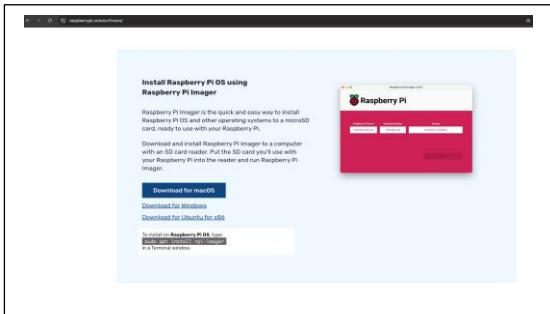
GND to the GND pin on the Raspberry Pi.

TRIG to GPIO Pin 23 on the Raspberry Pi.

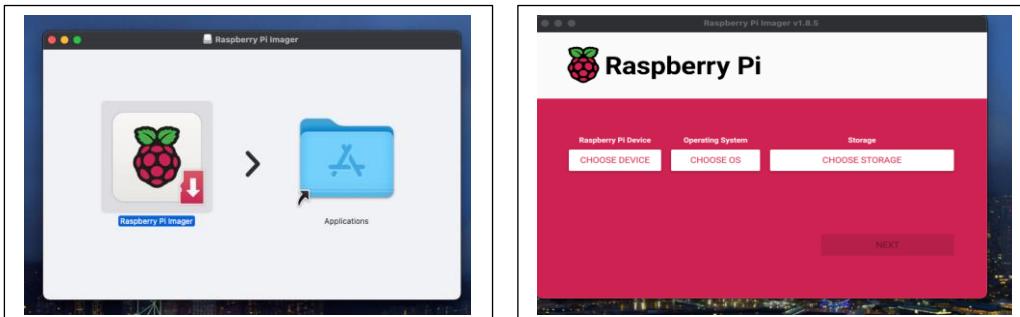
ECHO to GPIO Pin 24 on the Raspberry Pi.

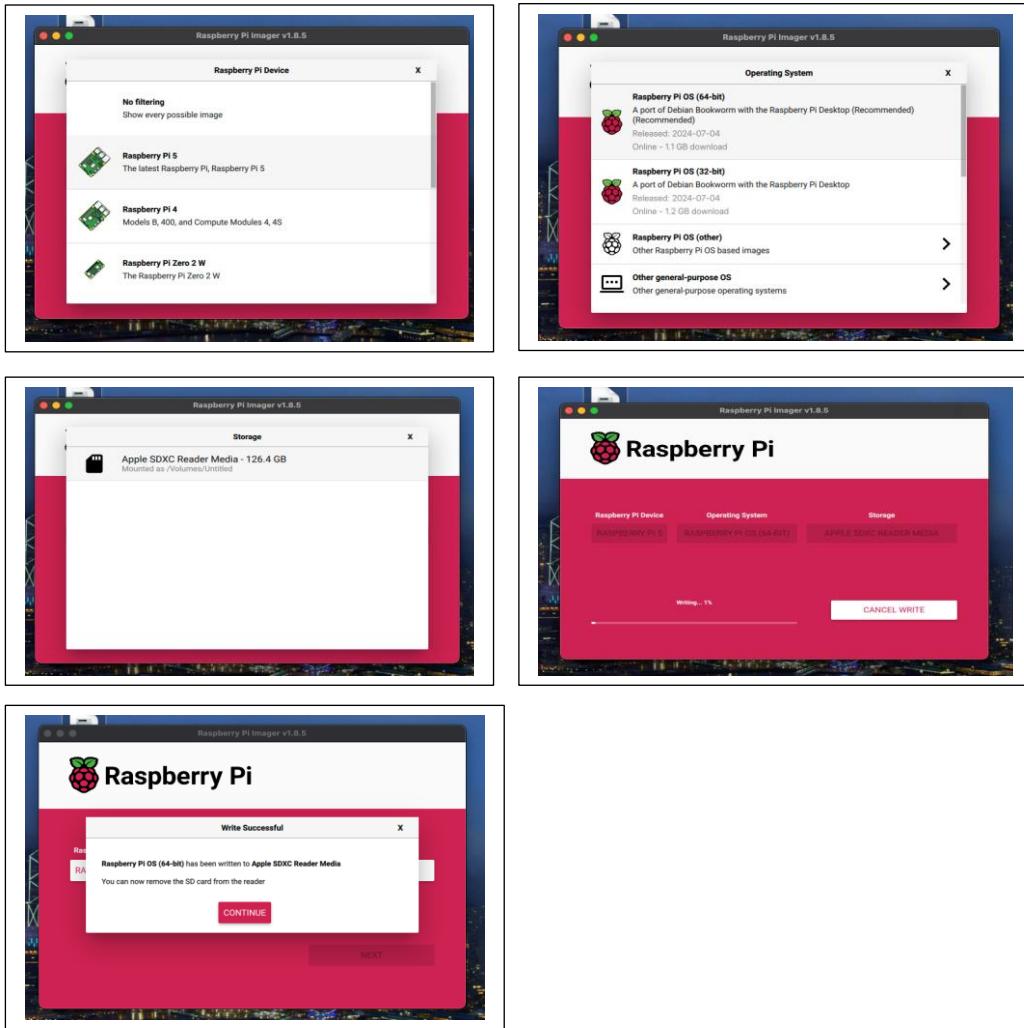
The Raspberry Pi GPIO pins operate at 3.3V, but the sensor operates at 5V. To protect the GPIO pins, I used a 1kΩ resistor and a 2kΩ resistor as a voltage divider for the ECHO pin to step the voltage down to a safe 3.3V before connecting it to the Raspberry Pi

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

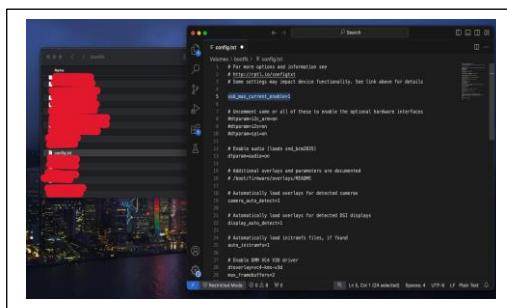


- b.) Ran the Raspberry Pi Imager to install the OS onto an external SD card.

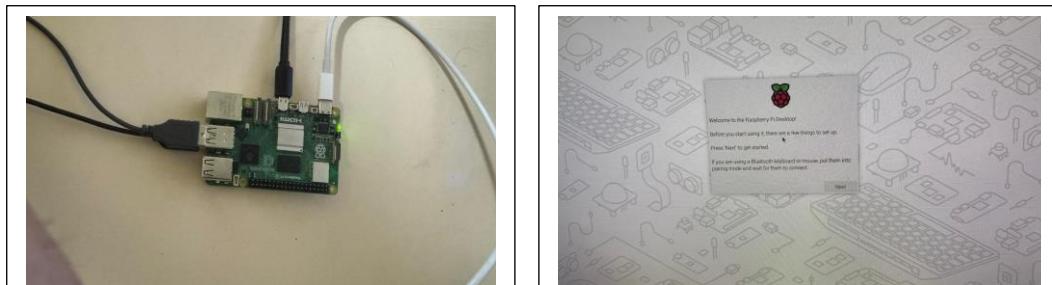


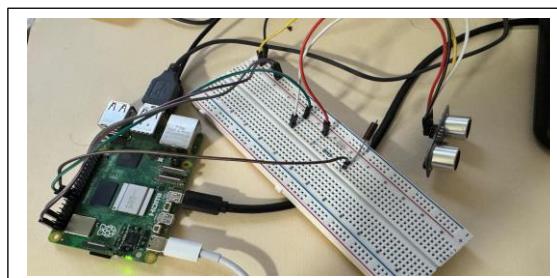
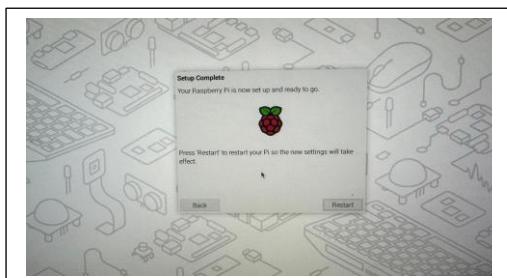
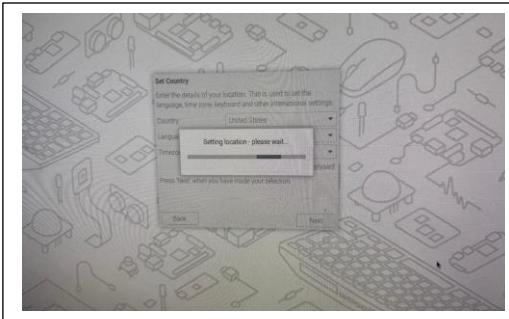


c.) Updated config.txt to enable the Pi to boot from the SD card.

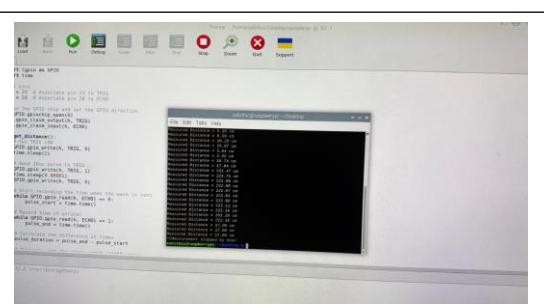
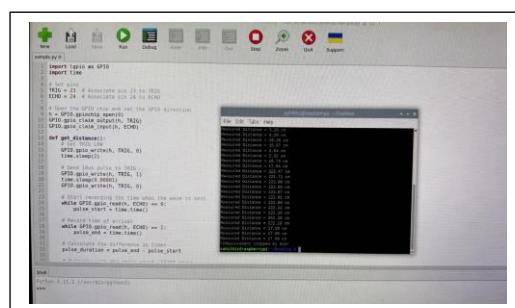


d.) Hardware Setup: Set up the hardware by connecting the Raspberry Pi with the ultrasonic sensor using a breadboard.

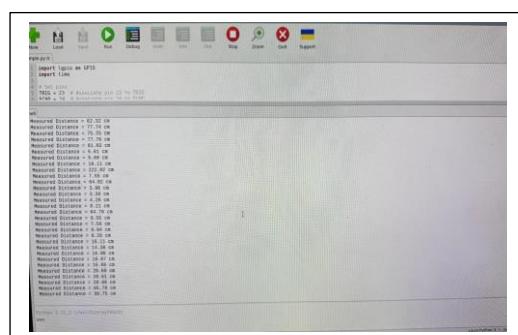




e.) Basic Sensor Functionality: Developed and ran a sample Python script to measure the distance between the sensor and an object for every 2 seconds.



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.

awsiot sdk: AWS IoT SDK to manage device communication.

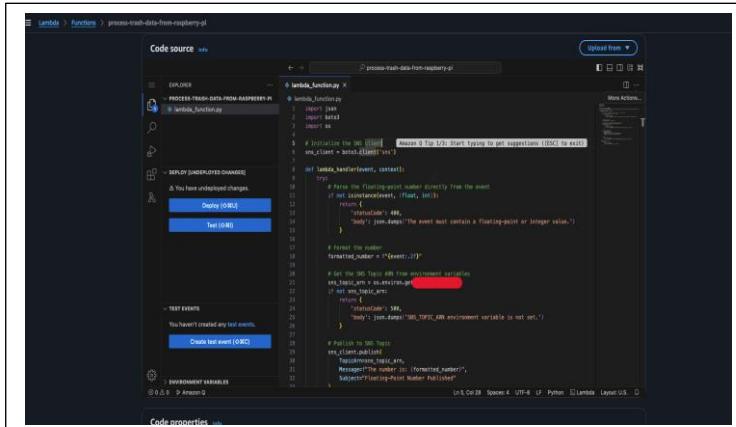
Since newer Raspberry Pi models may restricted direct package installations, I create a new Python environment, activated it and ran a sample script provided by AWS (named start.sh). This script automatically installed all necessary packages and dependencies.

```
./start.sh
Running ./start.sh...
Installing collected packages: paho-mqtt
  DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 won't be maintained after that date. A future version of pip will drop support for Python 2.7.
  Using legacy 'setup.py install' for paho-mqtt, since it does not have a 'pyproject.toml' or 'setup.cfg'.
  Running setup.py install for paho-mqtt ... done
Successfully installed paho-mqtt==1.6.0-py3

Running pub/sub sample application...
Connecting to ada-test-00000000000000000000000000000000 with client ID "basicPubSub"...
Connected!
Subscribing to topic "ada/test/python"...
Receiving message from topic "ada/test/python": "Hello World!" [1]
Publishing message to topic "ada/test/python": "Hello World!" [1]
Receiving message from topic "ada/test/python": "Hello World!" [2]
Publishing message to topic "ada/test/python": "Hello World!" [2]
Receiving message from topic "ada/test/python": "Hello World!" [3]
Publishing message to topic "ada/test/python": "Hello World!" [3]
Receiving message from topic "ada/test/python": "Hello World!" [4]
Publishing message to topic "ada/test/python": "Hello World!" [4]
Receiving message from topic "ada/test/python": "Hello World!" [5]
Publishing message to topic "ada/test/python": "Hello World!" [5]
Receiving message from topic "ada/test/python": "Hello World!" [6]
Publishing message to topic "ada/test/python": "Hello World!" [6]
Receiving message from topic "ada/test/python": "Hello World!" [7]
Receiving message from topic "ada/test/python": "Hello World!" [7]
[Truncated]
    sleep(1)
  )
done
sleep(1)
done
done@raspberrypi:~/aws-iot-startup$
```

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:

$$\text{Distance (cm)} = (\text{Time Difference} \times 34300) / 2$$

(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).

```

raspberrypi:~/Desktop/CSSE32FinalProjectFiles$ python3 distance_sensor.py
Starting distance measurements...
Measured Distance = 37.21 cm
Measurement stopped by User
Cleaned up GPIO resources
raspberrypi:~/Desktop/CSSE32FinalProjectFiles$ python3 distance_sensor.py
Starting distance measurements...
Measured Distance = 37.21 cm
Measurement stopped by User
Cleaned up GPIO resources
raspberrypi:~/Desktop/CSSE32FinalProjectFiles$ python3 distance_sensor.py
Starting distance measurements...
Measured Distance = 18.97 cm

```

```

sahithic@raspberrypi:~/Desktop/CSSE32FinalProjectFiles$ python3 distance_sensor.py
Starting distance measurements...
Measured Distance = 18.37 cm
Measured Distance = 17.69 cm
Measured Distance = 8.95 cm
Measurement stopped by User
Cleaned up GPIO resources
sahithic@raspberrypi:~/Desktop/CSSE32FinalProjectFiles$ 

```

5. Writing the Publishing Script

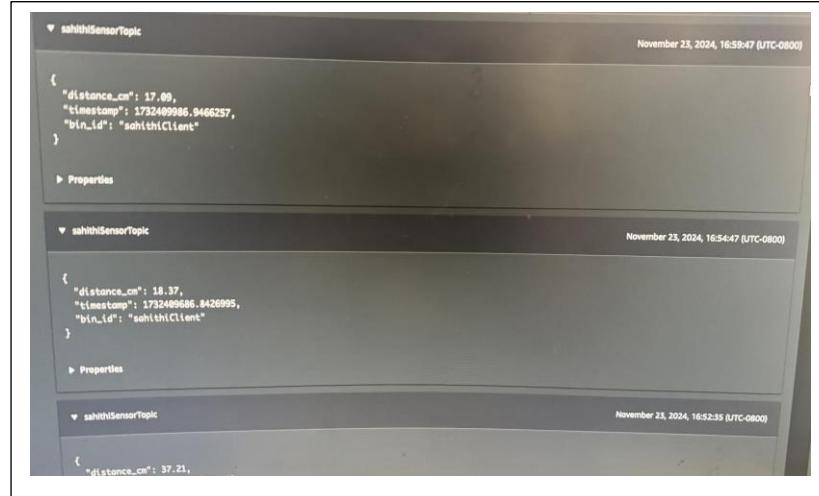
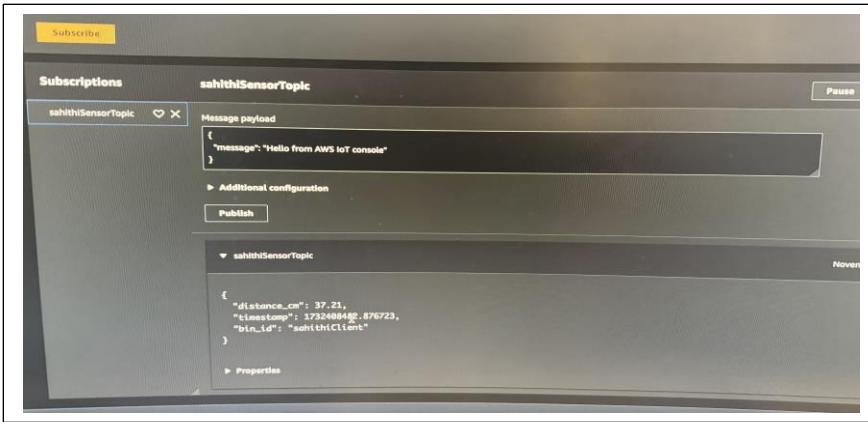
Now, I create a separate Python script named aws_publisher.py to handle data transmission to AWS IoT Core. This script reads the JSON file and extracts the most recent distance and timestamp data and publishes it to AWS IoT Core. This data is sent to AWS lambda for further processing.

```

(aws-iot-env) sahithic@raspberrypi:~/Desktop/CSSE32FinalProjectFiles$ python3 aws_publisher.py
Connecting to AWS IoT Core at ...
Connected to AWS IoT Core!
Starting AWS IoT publisher...
Published measurement: {'distance_cm': 37.21, 'timestamp': 1732400482.876723, 'bin_id': 'sahithicClient'}
AWSIoTPublisher stopped by user
Disconnected from AWS IoT Core
(aws-iot-env) sahithic@raspberrypi:~/Desktop/CSSE32FinalProjectFiles$ python3 aws_publisher.py
Connecting to AWS IoT Core at ...
Connected to AWS IoT Core!
Starting AWS IoT publisher...
Published measurement: {'distance_cm': 37.21, 'timestamp': 1732400482.876723, 'bin_id': 'sahithicClient'}
Published measurement: {'distance_cm': 18.97, 'timestamp': 1732400686.8426995, 'bin_id': 'sahithicClient'}

```

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.

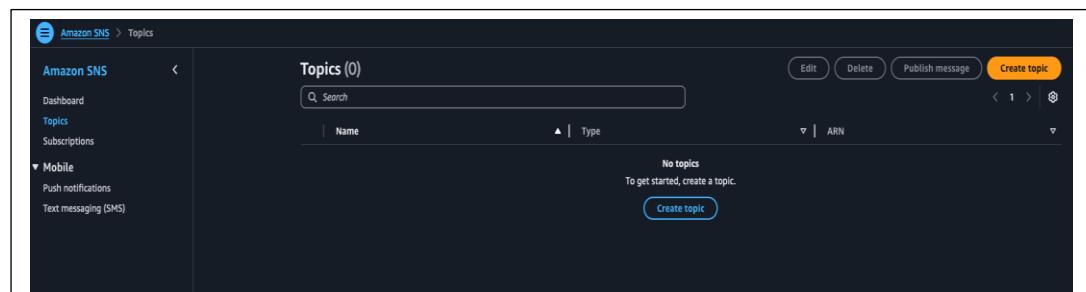
6. Setting up AWS IOT core: (already done during HW1 for CSS532)

(Note: Already done creating AWS account, setting up thing for HW1)

I created a new policy specifically for subscribing to a topic named “mySensorTopic”. Using the same Python script for HW1, I successfully subscribed to the topic by executing the script through the terminal. After configuring the policy, I used the AWS IoT console to subscribe to the same topic and verify that my device could successfully publish messages to it. This validated the functionality of both publishing and subscribing, ensuring seamless communication between my device and AWS IoT.

7. Setting Up Amazon SNS (Simple Notification Service):

I navigated to the SNS Console by opening the AWS Management Console and accessing the SNS service. Once there, I created a topic by selecting "Create Topic" and filling in the necessary details, such as the topic name and type (e.g., Standard).



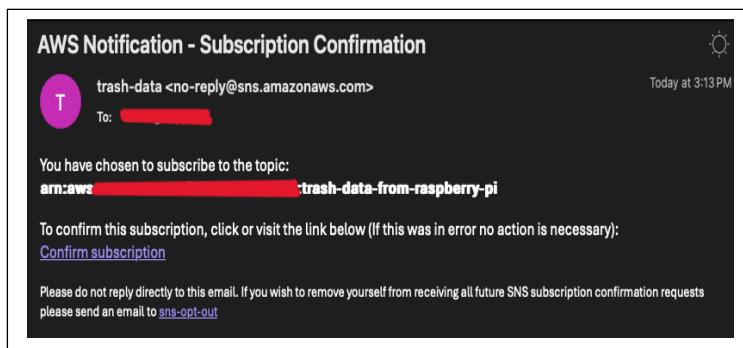
I assigned a unique name to the topic, such as trash-data-from-raspberry-pi, which reflects its purpose, and completed the topic creation process. The newly created topic was displayed in the list of available topics.

Next, I went to the Subscriptions tab under the created topic to set up a new subscription.

I selected the Email protocol as the method of notification. In the Endpoint field, I entered the recipient's email address and confirmed the creation of the subscription.

The first screenshot shows the 'Create subscription' page with the following fields: Topic ARN: arn:aws:sns:region:account-id:trash-data-from-raspberry-pi; Protocol: Email; Endpoint: recipient@example.com. A note says: 'After your subscription is created, you must confirm it.' The second screenshot shows the 'Subscription details' page with the same information, plus a green success message: 'Subscription to trash-data-from-raspberry-pi created successfully. The ARN of the subscription is arn:aws:sns:region:account-id:trash-data-from-raspberry-pi'. The status is listed as 'Pending confirmation'.

An email was automatically sent to the provided address for confirmation. Upon opening the email, I clicked the confirmation link, which validated the subscription and updated its status to Confirmed in the SNS console.



8. 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.

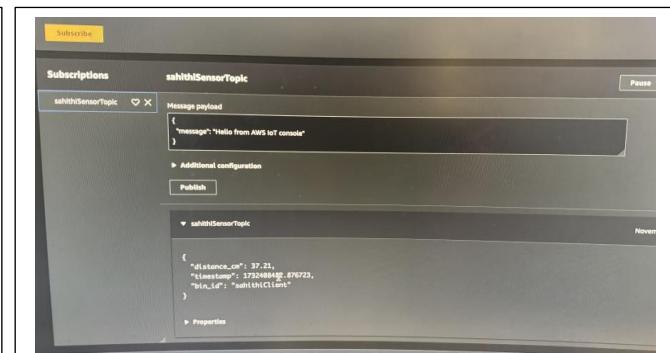
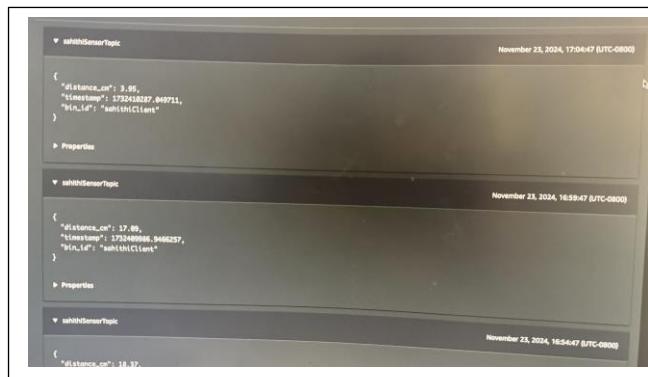
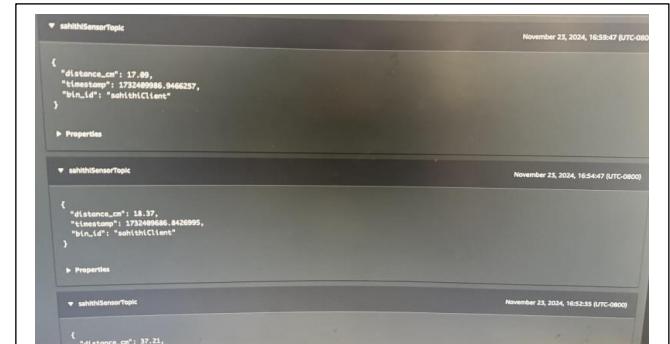
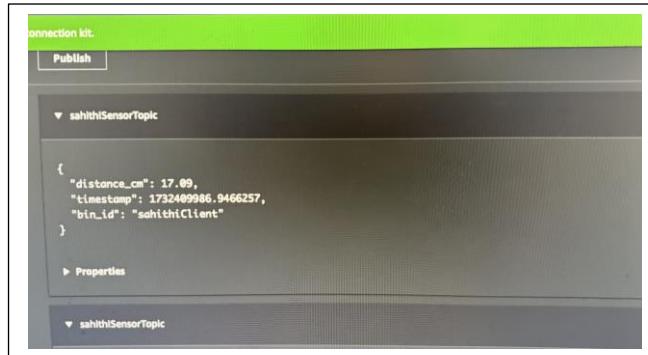
Then, I ran the publishing script (aws_publisher.py) to verify that the data from the JSON file is correctly transmitted to AWS IoT Core.

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.

```
sahithic@raspberrypi:~/Desktop/CS5532Final/ProjectFiles $ python3 distance_sensor.py
Starting distance measurements...
Measured Distance = 18.37 cm
Measured Distance = 17.69 cm
Measured Distance = 3.95 cm
ACMeasurement stopped by User
Cleaned up GPIO resources
sahithic@raspberrypi:~/Desktop/CS5532Final/ProjectFiles $
```

```
(aws-iot-env) sahithic@raspberrypi:~/Desktop/CS5532Final/ProjectFiles $ python3 aws_publisher.py
Connecting to AWS IoT Core at [REDACTED]
Connected to AWS IoT Core!
Starting AWS IoT publisher...
Published measurement: {"distance_cm": 37.21, "timestamp": 1732488482.876723, "pin_id": "sahithiClient"}
Published measurement: {"distance_cm": 18.37, "timestamp": 1732488686.8426995, "pin_id": "sahithiClient"}
Published measurement: {"distance_cm": 17.69, "timestamp": 1732488686.8486257, "pin_id": "sahithiClient"}
```



9. Setting Up the AWS Lambda Function:

Following this, I navigated to the AWS Lambda Console to create a new Lambda function. I chose "Create Function", selected Author from Scratch, and provided a descriptive name for the function, such as process-trash-data-from-raspberry-pi. I specified the runtime environment as Python 3.13 and assigned an appropriate IAM role with basic execution permissions to the function.

The image contains three screenshots of the AWS Lambda console interface:

- Screenshot 1: Functions List**
Shows the Lambda > Functions page with a list of 4 functions. The search bar is empty, and the filters are set to 'Function name' and 'Description'. The table includes columns for 'Package type', 'Runtime', and 'Last modified'. A 'Create function' button is visible in the top right.
- Screenshot 2: Create Function Wizard**
Shows the 'Create function' wizard step. It starts with a choice between 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. The 'Basic information' step is shown, where the function name 'process-trash-data-from-raspberry-pi' is entered. Below it, the 'Runtime' is set to 'Python 3.13' and 'Architecture' to 'x86_64'. The 'Permissions' section indicates a default execution role will be created. Other sections like 'Additional Configurations' are partially visible.
- Screenshot 3: Function Overview**
Shows the 'process-trash-data-from-raspberry-pi' function details. A success message at the top says 'Successfully created the function process-trash-data-from-raspberry-pi.' The 'Function overview' tab is active, showing the function name, layers (0), and triggers (0). The 'Code' tab is selected at the bottom.

After creating the function, I updated its permissions by navigating to the Configuration tab in the Lambda Console and selecting the IAM role linked to the function.

The screenshot shows the AWS Lambda Configuration page for a function named "process-trash-data-from-raspberry-pi". The "Execution role" tab is selected. The role name is "process-trash-data-from-raspberry-pi-role". The "Resource summary" section shows permissions for Amazon CloudWatch Logs, allowing CreateLogGroup, CreateLogStream, and PutLogEvents actions. The "By resource" tab is selected, showing two resources: "arn:aws:logs:us-west-2:log-group:/aws/lambda/process-trash-data-from-raspberry-pi" and "arn:aws:logs:us-west-2:log-group:/aws/lambda/process-trash-data-from-raspberry-pi:*". Below this, it states that Lambda obtained this information from the following policy statements:

- Managed policy AWSLambdaBasicExecutionRole
- Managed policy AWSLambdaBasicExecutionRole

Within the IAM policy editor, I added an inline policy to grant the Lambda function the necessary permissions to publish messages to the SNS topic.

The screenshot shows the IAM Role Permissions page for the role "process-trash-data-from-raspberry-pi-role". The "Permissions" tab is selected, showing one inline policy named "AWSLambdaBasicExecutionRole". The ARN of the policy is "arn:aws:iam::[REDACTED]:role/service-role/process-trash-data-from-raspberry-pi-role-AWSLambdaBasicExecutionRole".

This policy included the sns:Publish action and specified the ARN of the SNS topic created earlier. Once the policy was saved, it was successfully attached to the Lambda function.

The screenshot shows the AWS IAM Policy Editor. The "Policy editor" section displays the following JSON policy:

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Sid": "Statement1",  
6       "Effect": "Allow",  
7       "Action": "sns:Publish",  
8       "Resource": "arn:aws:sns:[REDACTED]:trash-data-from-raspberry-pi"  
9     }  
10   ]  
11 }
```

The "Actions" sidebar on the right lists services and actions available for attachment, including SNS, AMP, API Gateway, API Gateway V2, ARC Zonal Shift, ASC, Access Analyzer, and Account.

The screenshot shows the AWS IAM 'Policy publish-to-sns' details page. At the top, a green banner indicates the policy has been created. Below it, the ARN is listed as `arn:aws:iam::[REDACTED]::role/service-role/process-trash-data-from-raspberry-pi-role`. The maximum session duration is set to 1 hour. The 'Permissions' tab is selected, showing two attached policies: `AWSLambdaBasicExecutionRole` (Customer managed) and `publish-to-sns` (Customer inline). A 'Permissions boundary (not set)' section is also present.

I then wrote the Lambda function's code, which included logic to publish alert messages to the SNS topic.

The screenshot shows the AWS Lambda function code editor for the function `process-trash-data-from-raspberry-pi`. The code is written in Python and includes logic to handle events and publish messages to an SNS topic. The code editor interface shows the function name, deployment stage, and various configuration tabs like 'Code source', 'Environment variables', and 'Code properties'.

To make the function dynamic and maintainable, I used environment variables to store the SNS topic ARN.

The screenshot shows the 'Edit environment variables' configuration page for the Lambda function. It lists a single environment variable named `SNS_TOPIC_ARN` with the value `arn:aws:sns:[REDACTED]:trash-data-from-raspberry-pi`. There are buttons for 'Add environment variable' and 'Encryption configuration', along with 'Cancel' and 'Save' buttons at the bottom.

After deploying the code, I tested the Lambda function using the Test option available in the Lambda console. The function executed successfully, sending messages to the SNS topic.

The screenshot shows the AWS Lambda Test console interface. On the left, there's a sidebar with sections for 'TEST EVENTS' (containing 'Unsaved test events' and 'Private saved events' with one entry named 'Test1'), 'ENVIRONMENT VARIABLES' (with a gear icon), and footer links for 'Amazon Q' and other AWS services. The main area has tabs for 'PROBLEMS', 'OUTPUT' (which is selected), 'CODE REFERENCE LOG', and 'TERMINAL'. The 'OUTPUT' tab displays the following text:

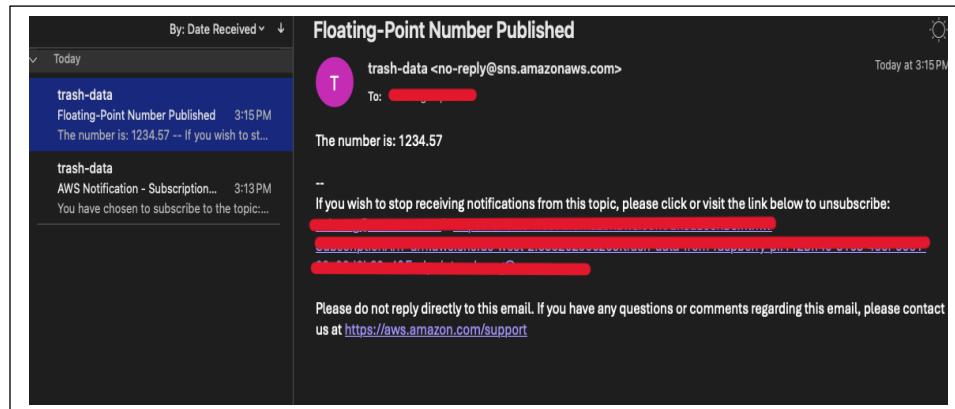
```
Status: Succeeded
Test Event Name: Test1

Response:
{
  "statusCode": 200,
  "body": "\"Number 1234.57 published to SNS topic.\""
}

Function Logs:
START RequestId: [REDACTED] Version: $LATEST
Ln 21, Col 56 Spaces: 4 UTF-8 LF Python Lambda Layout: U.S.
```

At the bottom, it shows 'Ln 21, Col 56 Spaces: 4 UTF-8 LF Python Lambda Layout: U.S.'

I verified the test results by checking that an email notification was received at the subscribed address.



10. Setting Up AWS IoT Core Rules:

With SNS and Lambda configured, I proceeded to the AWS IoT Core Console to set up a rule for routing messages from the IoT topic to the Lambda function.

```
{
  "Effect": "Allow",
  "Action": "iot:Subscribe",
  "Resource": [
    "arn:aws:iot:[REDACTED]:topicfilter/sahithiSensorTopic"
  ],
}
```

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Connect",
    "iot:Publish",
    "iot:Receive"
  ],
  "Resource": [
    "arn:aws:iot:[REDACTED]:topic/sahithiSensorTopic"
  ]
}
```

I defined a SQL statement within the IoT Core rules engine to filter and route data. For instance, I wrote a statement to select all data from the "mysensorTopic".

Step 1
specify rule properties

Step 2
Configure SQL statement

Step 3
Attach rule actions

Step 4
Review and create

Configure SQL statement Info

Add a simplified SQL syntax to filter messages received on an MQTT topic and push the data elsewhere.

SQL statement Info

SQL version
The version of the SQL rules engine to use when evaluating the rule.
2016-03-23

SQL statement
Enter a SQL statement using the following: `SELECT <Attributes> FROM <Topic Filter> WHERE <Condition>`. For example: `SELECT temperature FROM IoTTopic WHERE temperature > 50`. To learn more, see [AWS IoT SQL Reference](#).

```
1 SELECT * FROM 'sahithiSensorTopic'
```

Step 2: SQL statement

SQL statement

SQL version
2016-03-23

SQL query
`SELECT * FROM 'sahithiSensorTopic'`

Step 3: Rule actions

Actions

Lambda
Send a message to a Lambda function

Lambda function
`arn:aws:lambda:us-east-1:123456789012:function:process-trash-data-from-raspberry-pi`

Lambda function version
`$LATEST`

Error action

No error action

Cancel Previous Create

Successfully created rule RouteToProcessTrashDataLambda.

AWS IoT > Message routing > Rules > RouteToProcessTrashDataLambda

RouteToProcessTrashDataLambda Info

Details

Description
-

I created a new IoT rule, assigned it a descriptive name “RouteToProcessTrashDataLambda” and linked the Lambda function as the action to execute when the rule was triggered.

process-trash-data-from-raspberry-pi

Throttle Copy ARN Actions

Function overview Info

Diagram Template

process-trash-data-from-raspberry-pi

Layers (0)

AWS IoT Amazon SNS

+ Add trigger + Add destination

Export to Infrastructure Composer Download

Description
-

Last modified
31 minutes ago

Function ARN
`arn:aws:lambda:us-east-1:123456789012:function:process-trash-data-from-raspberry-pi`

Function URL Info
-

Code Test Monitor Configuration Aliases Versions

General configuration Triggers (1) Info

Triggers Find triggers Fix errors Edit Delete Add trigger

Trigger AWS IoT:RouteToProcessTrashDataLambda

arn:aws:iot:us-east-1:123456789012:rule/RouteToProcessTrashDataLambda

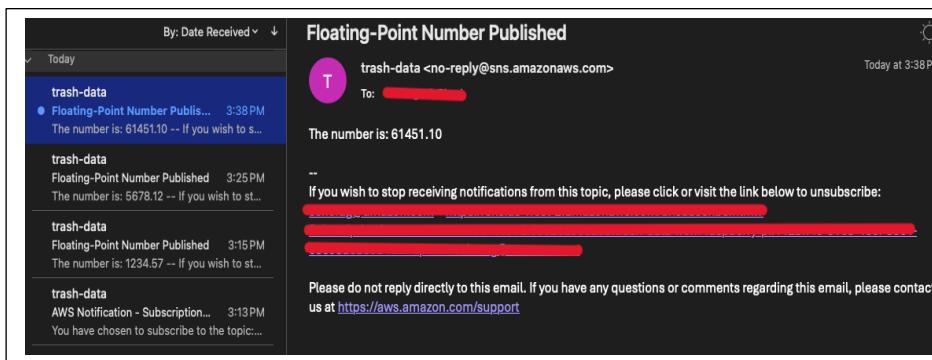
Details

After successfully creating the IoT rule, I tested the setup by publishing test data to the Topic using a Python script.

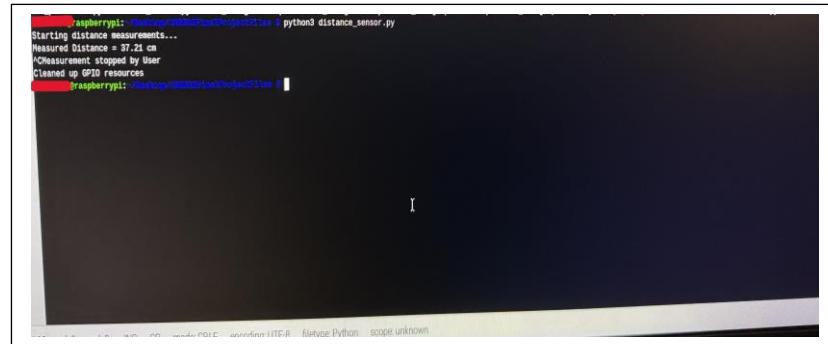


```
python3 testFloatNumbers.py
Connecting to endpoint [REDACTED] with client ID 'sahithiClient'...
Connected!
Subscribing to topic 'sahithiSensorTopic'...
Subscribed with QoS_AT_LEAST_ONCE
Generated number: 61451.10
Publishing 61451.1 to topic 'sahithiSensorTopic'
Received message from topic 'sahithiSensorTopic': b'61451.1'
Didn't receive numbers
Message successfully sent and received.
Disconnecting...
Disconnected!
```

I verified that the IoT Core rule routed the data to the Lambda function and triggered an SNS notification. Additionally, I validated that all components, including email notifications, worked as intended by monitoring their responses in real time.



Finally, I ran the original python script, starting with publishing real sensor data to AWS IoT Core from a Python script.

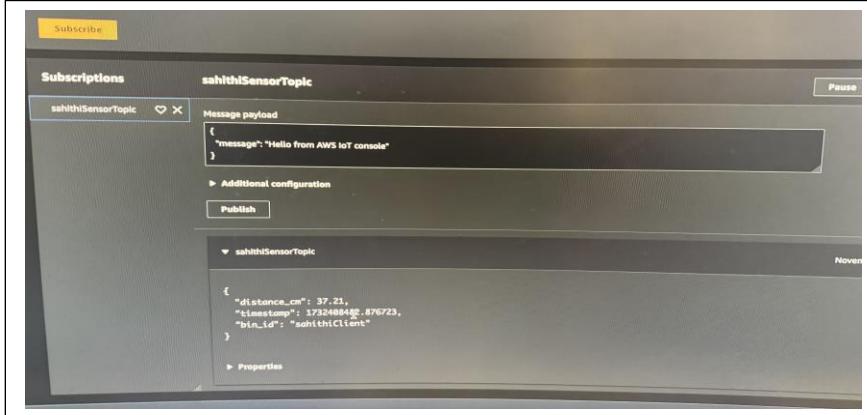


```
raspberrypi: ~$ cd ~/Desktop/05550FinalProjectFiles $ python3 distance_sensor.py
Starting distance measurement...
Measured Distance = 37.23 cm
Measurement stopped by User
Cleaned up GPIO resources
raspberrypi: ~$
```

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

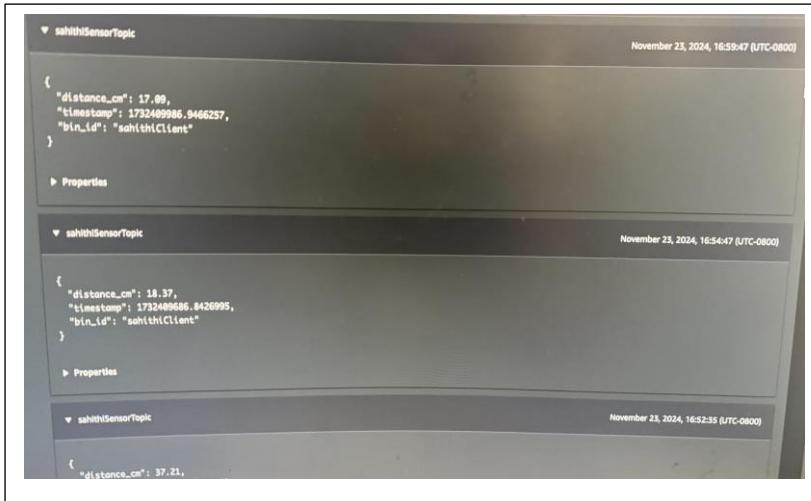


```
(aws-iot-env) sahithi@raspberrypi: ~$ cd ~/Desktop/05550FinalProjectFiles $ python3 aws_publisher.py
Connecting to AWS IoT Core at [REDACTED]
Connected to AWS IoT Core!
Starting AWS IoT publisher...
Published measurement: { 'distance_cm': 37.23, 'timestamp': '1793468482.879729', 'bin_id': 'sahithiClient' }
#Publisher stopped by user
Disconnected from AWS IoT Core
(sahithi-env) sahithi@raspberrypi: ~$ cd ~/Desktop/05550FinalProjectFiles $ python3 aws_publisher.py
Connecting to AWS IoT Core at [REDACTED]
Connected to AWS IoT Core!
Starting AWS IoT publisher...
Published measurement: { 'distance_cm': 37.23, 'timestamp': '1793468482.879729', 'bin_id': 'sahithiClient' }
```



```
sahithic@raspberrypi:~/Desktop/CSS532FinalProjectFiles $ python3 distance_sensor.py
Starting distance measurements...
Measured Distance = 18.37 cm
Measured Distance = 17.09 cm
Measured Distance = 3.95 cm
Measurement stopped by User
Cleaned up GPIO resources
sahithic@raspberrypi:~/Desktop/CSS532FinalProjectFiles $
```

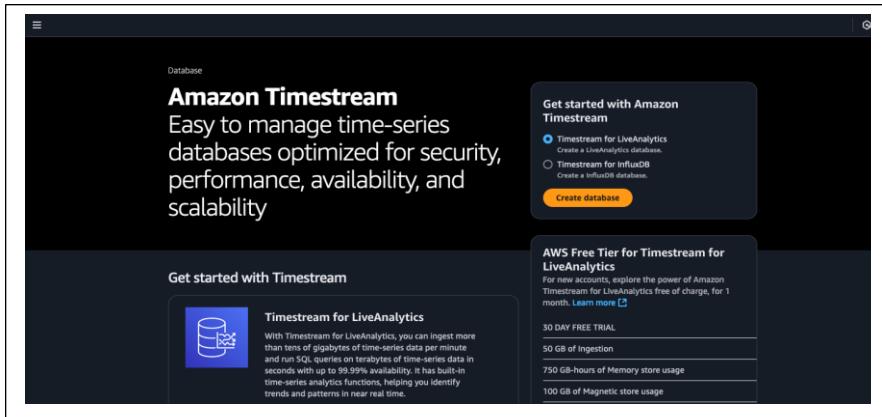
```
(aws-iot-env) sahithic@raspberrypi:~/Desktop/CSS532FinalProjectFiles $ python3 aws_publisher.py
Connecting to AWS IoT Core at [REDACTED]
Connected to AWS IoT Core!
Starting AWS IoT publisher...
Published measurement: {'distance_cm': 37.21, 'timestamp': 1732488482.876723, 'bin_id': 'sahithiClient'}
Published measurement: {'distance_cm': 18.37, 'timestamp': 1732409686.842695, 'bin_id': 'sahithiClient'}
Published measurement: {'distance_cm': 17.09, 'timestamp': 1732409686.9466257, 'bin_id': 'sahithiClient'}
```



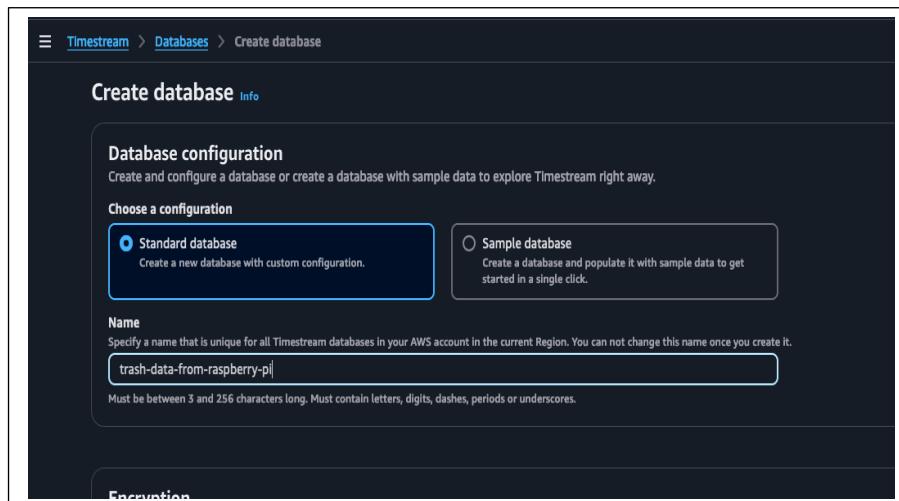
All notifications were successfully delivered to the email subscribers, ensuring seamless integration and functionality across the system.

11. Setting up TimeStream database:

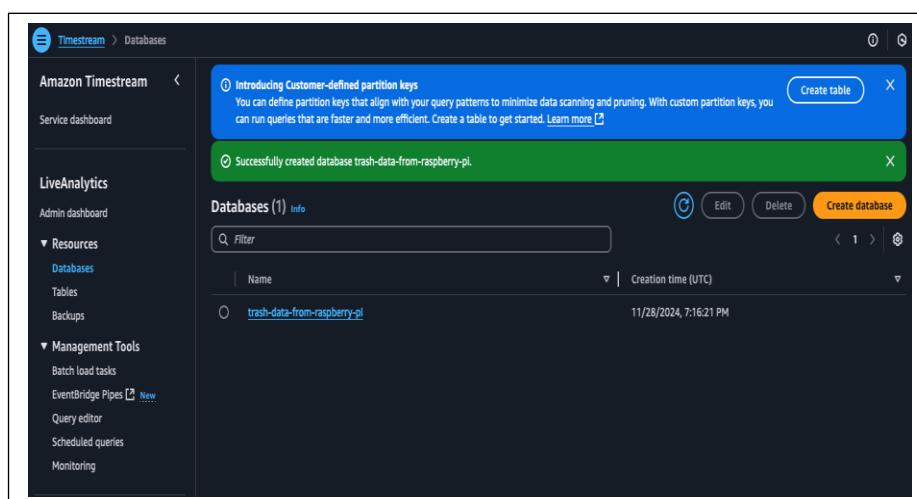
To set up and utilize Amazon Timestream for real-time analytics, I began by navigating to the Timestream Console on the AWS Management Console.



I selected the option to create a new database, opting for the "Standard database" type to suit the project's needs. I named it as "trash-data-from-raspberry-pi".



Upon entering the necessary details, I finalized the setup, receiving a confirmation message indicating the successful creation of the database.



After the database creation, I navigated to the database's page to view its details. In the Tables tab, I initially observed that no tables had been created yet, confirming the need for further configuration.

This screenshot shows the 'trash-data-from-raspberry-pi' database summary page in the Amazon Timestream console. The left sidebar includes links for Service dashboard, LiveAnalytics, Resources (Databases, Tables, Backups), Management Tools (Batch load tasks, EventBridge Pipes, Query editor, Scheduled queries, Monitoring), and InfluxDB. The main area displays the database name, ARN, and creation time. The 'Tables' tab is selected, showing 0 tables. There are buttons for Create backup, Create scheduled query, Edit, Delete, and Create table.

To proceed, I created a new table by selecting "Create table," and named it as "distance-data" to clearly represent the data it would store. I chose the default settings, including default partitioning, as they are well-suited for the typical use cases of this project.

This screenshot shows the same database summary page as the previous one, but now with a single table listed under the 'Tables' tab. The table is named 'distance-data'. The other tabs (Monitoring, Tags) are visible at the bottom.

This screenshot shows the 'Create table' configuration page for the 'distance-data' table. It includes sections for Data retention, Magnetic store retention, Magnetic Storage Writes, and Backup settings. The 'Data retention' section specifies memory store retention for 12 hours and magnetic store retention for 10 years. The 'Magnetic store retention' section specifies a value of 10 years. The 'Magnetic Storage Writes' section has an unchecked checkbox for 'Enable magnetic storage writes'. The 'Backup settings' section indicates integration with AWS Backup and has an unchecked checkbox for 'Turn on backups for this table'.

The table creation process was completed, and the system confirmed the successful creation of the table.

A screenshot of the Amazon Timestream Tables page. On the left, there's a sidebar with options like 'Amazon Timestream', 'Service dashboard', 'LiveAnalytics', 'Admin dashboard', 'Resources', 'Databases', 'Tables' (which is selected), 'Backups', 'Management Tools', 'Batch load tasks', 'EventBridge Pipes', and 'Query editor'. The main area shows a table with one row. The table has columns for 'Table name' (set to 'distance-data'), 'Database' (set to 'trash-data-from-raspberry-pi'), and 'Creation time (UTC)' (set to '11/28/2024, 7:24:48 PM'). Above the table, there are two notifications: a blue one about 'Introducing Customer-defined partition keys' and a green one stating 'Successfully created table distance-data.' Below the table are buttons for 'Create backup', 'Create scheduled query', 'Edit', 'Delete', and 'Create table'.

Once the Raspberry Pi started collecting waste level data using the ultrasonic sensor and publishing it to AWS IoT Core, this data was stored in the Timestream database. I wrote a SQL query and returned to the Query Editor and ran the query, before and after collecting the data from raspberry-pi.

A screenshot of the Query Editor. The top navigation bar includes 'Editor', 'Recent', 'Saved queries', and 'Sample queries'. The 'Editor' tab is selected. On the left, there's a 'Database' dropdown set to 'trash-data-from-raspberry-pi' and a 'Tables' section with one entry 'distance-data'. The main area contains a query editor window with the following content:

```
distance-data > FROM "trash-data-from-raspberry-pi"."distance-data"
```

Below the query editor are 'Save', 'Clear', and 'Run' buttons, and an 'Enable Insights' checkbox. At the bottom, the 'Table details' tab is active, showing the start time as '11:27:47 AM', a success status, '0 rows affected.', and the executed statement: 'SELECT * FROM "trash-data-from-raspberry-pi"."distance-data"'.

A screenshot of the Query Editor, identical to the previous one but showing the results of the query execution. The 'Query results' tab is now active, displaying the following output:

```
distance-data > SELECT * FROM "trash-data-from-raspberry-pi"."distance-data"
```

Below the query results are 'Save', 'Clear', and 'Run' buttons, and an 'Enable Insights' checkbox. At the bottom, the 'Table details' tab is active, showing the start time as '11:28:55 AM', a success status, '24 rows returned.', and the executed statement: 'SELECT * FROM "trash-data-from-raspberry-pi"."distance-data"'.

The results displayed records with details such as BinID, source, measure_name, timestamp and measure_value(distance_cm), verifying that the data was being collected and stored accurately.

Rows returned (24)					
BinID	Source	measure_name	time	measure_value::double	
bin1	Raspberry-Pi	trash_bin_remainining_capacity	2024-11-27 20:25:13.10 6000000	28.7	
bin1	Raspberry-Pi	trash_bin_remainining_capacity	2024-11-27 20:20:13.00 2000000	28.66	
bin1	Raspberry-Pi	trash_bin_remainining_capacity	2024-11-27 20:15:12.89 7000000	28.66	
bin1	Raspberry-Pi	trash_bin_remainining_capacity	2024-11-27 20:10:12.79 3000000	28.7	
bin1	Raspberry-Pi	trash_bin_remainining_capacity	2024-11-27 20:05:12.68 9000000	28.5	
bin1	Raspberry-Pi	trash_bin_remainining_capacity	2024-11-27 20:00:12.58 4000000	28.69	
bin1	Raspberry-Pi	trash_bin_remainining_capacity	2024-11-27 19:55:12.48 0000000	28.68	
bin1	Raspberry-Pi	trash_bin_remainining_capacity	2024-11-27 19:50:12.37 5000000	28.66	
		trash_bin_re	2024-11-27		

This process validated the successful setup and integration of Amazon Timestream for live analytics, enabling real-time monitoring and analysis of waste bin levels.



12. Setting up Graphana dashboard:

To set up and utilize Grafana for visualizing data stored in Amazon Timestream, I started by creating a new Grafana workspace in the AWS Management Console.

Specify workspace details

give a unique name to your workspace.

trash-data-visualization

Valid special characters include: ~, ^, *, _, ~-. Cannot contain non-ASCII characters or spaces. Max length of 255 characters.

Workspace description - optional

Grafana version

Select the Grafana version you want to use for this workspace.

10.4

▼ Tags - optional

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Currently not editing fields.

Add new tag

You can add up to 50 more tags.

Cancel Next

After navigating to the Grafana service, I selected the option to create a workspace, configuring the necessary workspace settings to align with the project's requirements and selected time-Stream as data source for Graphana dashboard.

Configure settings

Choose at least one authentication method.

AWS IAM Identity Center (successor to AWS SSO) Enabled

You can enable IAM Identity Center by creating a user. This new user does not automatically have access to the Grafana console. You will still need to assign this user later, once this workspace is created.

Security Assertion Markup Language (SAML)

You will need to complete additional steps to finish SAML configuration once this workspace is created.

Permission type

Service managed

We will automatically provision the permissions for you based on the AWS services you choose in the next step.

Customer managed

Manually create your own IAM role based on the suggested policies.

Outbound VPC connection - optional

You can optionally connect your Grafana workspace to Amazon Virtual Private Cloud (Amazon VPC). Connect your workspace to a VPC in the same AWS account and Region to access resources in that VPC without sending requests across the public Internet. Configuring the VPC will also create a new IAM service-linked role that allows Amazon Managed Grafana to read your VPC settings, to create connections to the VPC, and to delete those created connections.

Warning

Service managed permission settings

IAM permission access settings

Select how you would like to specify account access.

Current account

Use Grafana to monitor resources in your current account.

Organization

Use Grafana to monitor resources in your Organizational Units (OUs).

Data sources and notification channels - optional

Data sources

Selecting an AWS data source below creates an IAM role that enables Amazon Grafana access to those resources in your current account. It does not set up the selected service as a data source. Note that some resources must be tagged GrafanaDataSource to be accessible.

Data source name

- AWS IoT SiteWise
- AWS X-Ray
- Amazon CloudWatch
- Amazon OpenSearch Service
- Amazon Managed Service for Prometheus
- Amazon TimeStream
- Amazon Redshift
- Amazon Athena

Notification channels

Once the settings were finalized, the workspace creation was completed successfully, providing a dedicated URL for accessing the Grafana dashboard.

The screenshot shows the 'trash-data-visualization' workspace summary page. At the top, a green banner indicates 'Workspace trash-data-visualization successfully created.' Below the banner, the workspace name 'trash-data-visualization' is displayed. The 'Summary' tab is selected. Key details shown include:

- Description: [redacted]
- Date created: November 25, 2024 at 14:16 (UTC-00:00)
- IAM role: amaws-[redacted]AmazonGrafanaServiceRole
- Grafana workspace URL: [redacted].grafana-workspace.us-west-2.amazonaws.com
- Status: Active
- Grafana version: 10.4

Initially, no users were added to the workspace.

The screenshot shows the 'Assign new user or group' section of the user management interface. It includes a note about enabling AWS IAM Identity Center and a warning message: '⚠ Assign new users to the Grafana workspace so users can access the workspace URL.'

I navigated to the user management section and added a new user with the Viewer role.

The screenshot shows the 'Assign user' dialog. Under the 'Users (1)' tab, a single user is selected, indicated by a blue highlight. The 'Selected users and groups (1)' section shows the same user. At the bottom right, there is a large orange 'Assign users and groups' button.

The screenshot shows the 'AWS IAM Identity Center (successor to AWS SSO)' configuration page. Under the 'Assigned users' tab, a single user is listed with the role 'Viewer'. The 'User type' column shows 'Viewer'. At the bottom right, there is a 'Delete configuration' button.

After verifying the user's addition, I updated the role to Admin, which enabled me to make changes to the workspace. This update was reflected in the user management interface, confirming the role change.

This screenshot shows the AWS IAM Identity Center interface for a workspace named "trash-data-visualization". In the "Assigned users" tab, there is one user listed: "Viewer" (Full name redacted). A context menu is open for this user, with the "Make admin" option highlighted in blue. Other options include "Assign user", "Unassign user", "Make editor", and "Make viewer".

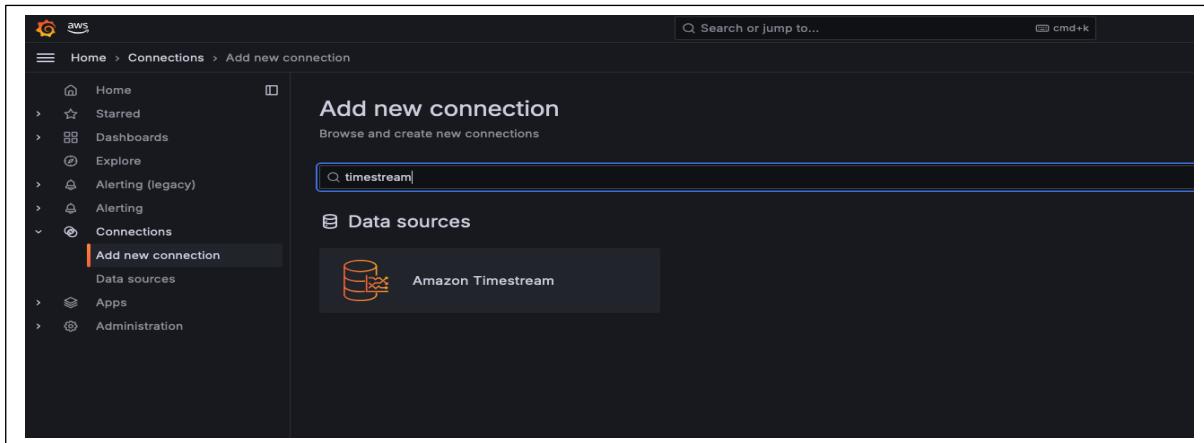
This screenshot shows the same AWS IAM Identity Center interface after the role update. The user "Viewer" is now listed with "User type" set to "Admin". The "Action" dropdown menu is visible on the right.

Next, I accessed the Grafana workspace URL and signed in using the designated username and password. After a successful login, the Grafana homepage became visible, marking the completion of the setup phase.

The top half of this image shows the "trash-data-visualization" workspace summary page in the Grafana interface, displaying the workspace URL and status as "Active". The bottom half shows the "Welcome to Amazon Managed Grafana" sign-in screen, where a user has logged in successfully.

The top half of this image shows the "Sign in" screen for the workspace, with the "Sign in" button highlighted. The bottom half shows the "Welcome to Amazon Managed Grafana" dashboard, featuring various data sources and visualization options.

I proceeded to add Amazon Timestream as the data source for the workspace.



However, I encountered a restriction—by default, the workspace lacked permission to manage plugins.

A screenshot of the 'Amazon Timestream' plugin configuration page. The left sidebar is identical to the previous screenshot. The main content area starts with an error message: 'You do not have permission to install this plugin.' Below this are tabs for 'Overview' (which is active) and 'Version history'. Under 'Overview', there's a section for 'Timestream Datasource' with the sub-instruction: 'The Timestream datasource plugin provides a support for Amazon Timestream. Add it as a data source, then you are ready to build'. There are also sections for 'Add the data source', 'Authentication', 'IAM policies', and a link to 'AWS authentication topic'.

A screenshot of the 'Workspace configuration options - new' tab in the Grafana settings. The top navigation bar includes 'Authentication', 'Data sources', 'Notification channels', 'Tags', 'Network access control', and 'Workspace configuration options - new' (which is highlighted). Below are three sections: 'Grafana alerting' (Info, Off), 'Plugin management - new' (Info, Off), and 'Network access control'. Each section has an 'Edit' button.

To address this, I enabled the Plugin Management feature within the workspace settings.

The screenshot shows the 'Workspace configuration options' page in Grafana. It includes sections for 'Grafana alerting' (with a note about Prometheus alerts) and 'Plugin management' (with a note about workspace admins). A 'Save changes' button is at the bottom right.

The screenshot shows the 'trash-data-visualization' workspace summary in Grafana. It displays basic information like the workspace URL (redacted), status (Active), and date created (November 25, 2024). A green banner at the top indicates the workspace was updated successfully.

Once this feature was activated, I returned to the console, where an option to install the Amazon Timestream plugin became available.

The screenshot shows the 'Connections' page in Grafana. The 'Data sources' section is selected. An 'Amazon Timestream' plugin card is visible, showing its version (2.9.11), source (Grafana Labs), and download count (8,364,684). A blue 'Install 2.9.11' button is at the top right of the card.

The screenshot shows the Grafana Marketplace page for the 'Amazon Timestream' plugin. At the top, there's a progress bar indicating 'Installing latest version of Amazon Timestream'. Below it, the plugin details are shown: 'Managed timeseries database from amazon', 'Website', and two tabs: 'Overview' (selected) and 'Version history'. A large central section is titled 'Timestream Datasource' with the sub-instruction: 'The Timestream datasource plugin provides a support for Amazon Timestream. Add it as a data source, then you are ready to build dashboards using timestream query results'. Below this is a 'Add the data source' section with a numbered list: 1. In the side menu under the Configuration link, click on Data Sources. 2. Click the Add data source button. 3. Select Timestream in the Time series databases section.

This screenshot shows the same 'Amazon Timestream' plugin page from the Grafana Marketplace, but with more content visible. It includes sections for 'Authentication' (with a note: 'For authentication options and configuration details, see AWS authentication topic.') and 'IAM policies'. The overall layout is identical to the first screenshot, with the addition of these two new sections.

After installing the plugin successfully, I added the Timestream database as a data source and tested the connection. The test confirmed that the database was integrated with Grafana.

The screenshot shows the 'grafana-timestream-datasource' configuration page in Grafana. The top navigation bar includes 'Settings' (selected), 'Dashboards', 'Permissions', and 'Insights'. The main area is titled 'Connection Details' and contains fields for 'Authentication Provider' (redacted), 'Assume Role ARN' (redacted), 'External ID' (set to 'External ID'), 'Endpoint' (redacted), and 'Default Region' (set to 'us-west-2'). Below this is the 'Timestream Details' section, which includes a note: 'Default values to be used as macros'. It lists 'Database' ('trash-data-from-raspberry-pi'), 'Table' ('distance-data'), and 'Measure' ('trash_bin_remaining_capacity'). At the bottom are 'Delete' and 'Save & test' buttons.

The screenshot shows the 'Timestream Details' configuration page. It displays the following settings:

Database	"trash-data-from-raspberry-pi"
Table	"distance-data"
Measure	trash_bin_remaining_capacity

A green success message box contains the text: "Connection success". Below it, a note says: "Next, you can start to visualize data by [building a dashboard](#), or by querying data in the [Explore view](#)". At the bottom are two buttons: "Delete" (pink) and "Save & test" (blue).

With the connection in place, I created a new dashboard, inputting the database details and a custom query to fetch relevant data.

The screenshot shows the 'New dashboard' creation interface. On the left is a sidebar with navigation links: Home, Starred, Dashboards (selected), Playlists, Snapshots, Library panels, Explore, Alerting (legacy), Alerting, Connections (selected), Add new connection, Data sources, Apps, and Administration. The main area has a central box with the heading 'Start your new dashboard by adding a visualization'. It includes a sub-instruction: 'Select a data source and then query and visualize your data with charts, stats and tables or create lists, markdowns and other widgets.' A large blue button labeled '+ Add visualization' is centered. Below this are three smaller boxes: 'Add a widget' (Create lists, markdowns and other widgets), 'Add a library panel' (Add visualizations that are shared with other dashboards), and 'Import a dashboard' (Import dashboard from file or grafana.com). A 'Last 6 hours' time range selector is at the top right.

The screenshot shows the 'Query' editor interface. At the top, there are tabs for 'Query' (1), 'Transform data' (0), and 'Alert' (0). Below is a header bar with 'Data source' set to 'grafana-timestream-datasource', a search icon, 'Query options' (MD = auto = 1187, Interval = 20s), and a 'Query inspector' tab. The main area is divided into sections: 'Macros' (Database: "trash-data-from-raspberry-pi", Table: "distance-data", Measure: "trash_bin_remaining_capacity"), 'Render' (Wait for all queries: off), 'Frames' (Format as: Table), and a code editor containing the query: 'SELECT * FROM "trash-data-from-raspberry-pi"."distance-data"'.

The dashboard successfully displayed the data in both graphical and table views, providing an interactive and informative visualization.

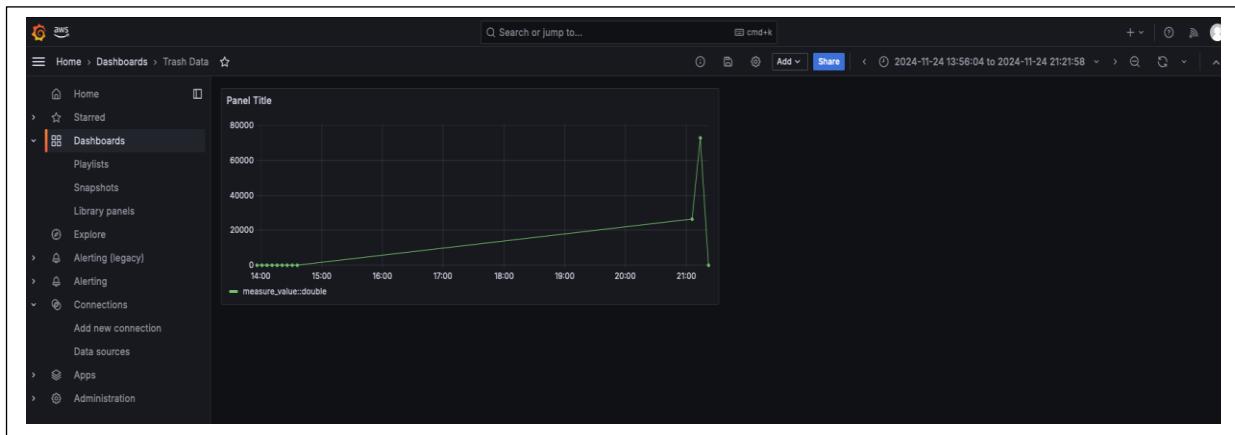


A table view showing detailed data for the "trash_bin_remaining_capacity" metric. The table has columns: BinID, Source, measure_name, time, and measure_value::double. The data shows multiple measurements taken by a Raspberry Pi device for a bin labeled "bin1". The values fluctuate between 2.36 and 73,119.

BinID	Source	measure_name	time	measure_value::double
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 21:05:45	26431
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 21:13:56	73119
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 13:56:04	29.5
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 14:01:04	30.8
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 14:06:04	34.0
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 14:11:05	2.36
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 14:16:05	7.54
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 14:21:05	2.97
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 14:26:05	9.93
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 14:31:05	9.92
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 14:36:05	9.91
bin1	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 21:21:58	32.3

Finally, I saved the newly created dashboard, ensuring the configuration was preserved for future use. This process established a seamless pipeline for monitoring and analysing the data collected by the smart waste management system using Grafana and Amazon Timestream.

A "Save dashboard" dialog box. It contains fields for "Title" (set to "Trash Data"), "Description" (empty), and "Folder" (set to "Dashboards"). At the bottom are "Cancel" and "Save" buttons.



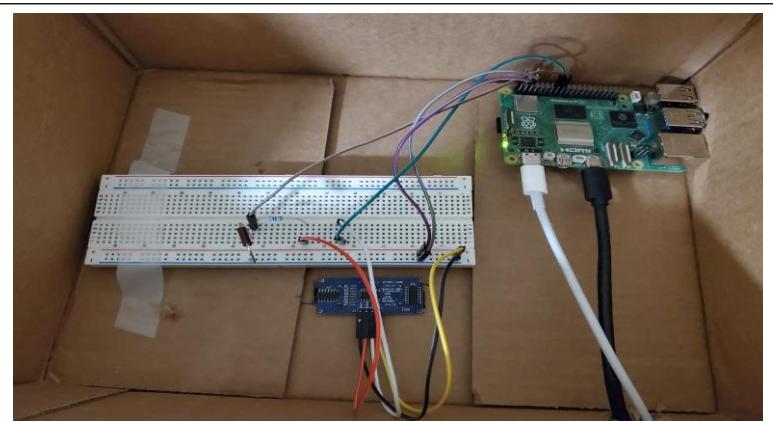
13. Setting Up Data Validation and Logging:

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.

14. 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.

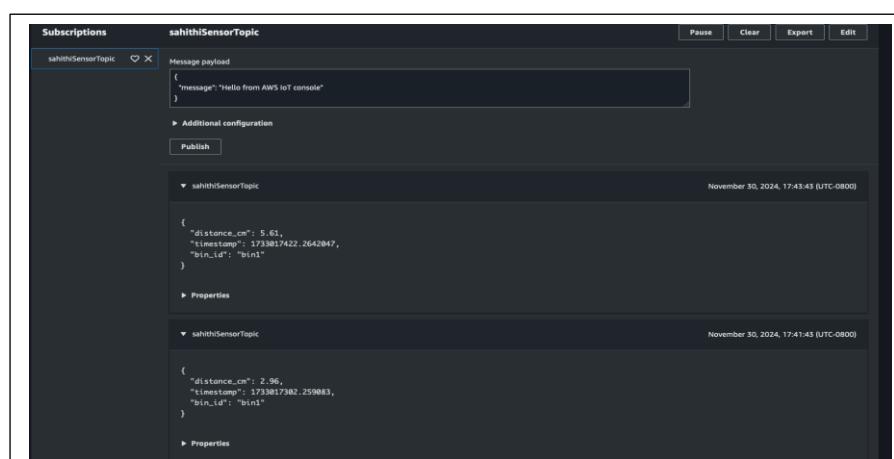


The testing involved running two Python scripts on the Raspberry Pi. The first script interfaced with the ultrasonic sensor, measured the distance to the waste level, and stored the data in a shared JSON file.

```
sahithic@raspberrypi:~/Desktop/CSS532FinalProjectFiles $ python3 distance_sensor1.py
Starting distance measurements...
Measured Distance = 28.69 cm
Measurement saved successfully.
Measured Distance = 16.07 cm
Measurement saved successfully.
Measured Distance = 14.73 cm
Measurement saved successfully.
Measured Distance = 6.25 cm
Measurement saved successfully.
Measured Distance = 3.95 cm
Measurement saved successfully.
Measured Distance = 8.22 cm
Measurement saved successfully.
Measured Distance = 2.06 cm
Measurement saved successfully.
Measured Distance = 3.29 cm
Measurement saved successfully.
Measured Distance = 2.63 cm
Measurement saved successfully.
Measured Distance = 9.59 cm
Measurement saved successfully.
Measured Distance = 2.96 cm
Measurement saved successfully.
Measured Distance = 5.61 cm
Measurement saved successfully.
Measured Distance = 17.40 cm
Measurement saved successfully.
Measured Distance = 13.70 cm
Measurement saved successfully.
Measured Distance = 16.71 cm
Measurement saved successfully.
Measured Distance = 27.23 cm
Measurement saved successfully.
Measured Distance = 29.54 cm
Measurement saved successfully.
^CMeasurement stopped by User
Cleaned up GPIO resources
```

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
```



The image consists of three vertically stacked screenshots from the AWS IoT Core console.

- Screenshot 1:** Shows the "Message payload" field containing the JSON object: { "message": "Hello from AWS IoT console" }. Below it, the "Publish" button is visible. The "Topic" dropdown is set to "sahithiSensorTopic".
- Screenshot 2:** Shows the published message details under the "sahithiSensorTopic" topic. The message content is identical to the one in Screenshot 1. The timestamp is November 30, 2024, 17:45:43 (UTC-0800).
- Screenshot 3:** Shows the published message details under the "sahithiSensorTopic" topic. The message content is identical to the ones in the previous screenshots. The timestamp is November 30, 2024, 17:49:45 (UTC-0800).
- Screenshot 4:** Shows the published message details under the "sahithiSensorTopic" topic. The message content is identical to the ones in the previous screenshots. The timestamp is November 30, 2024, 17:53:45 (UTC-0800).
- Screenshot 5:** Shows the published message details under the "sahithiSensorTopic" topic. The message content is identical to the ones in the previous screenshots. The timestamp is November 30, 2024, 17:51:43 (UTC-0800).

(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 17:23:42.21 1000000	55.41	
1.07	NORMAL	98.93	Raspberry-Pi	28.69	remaining_capacity_percentage	2024-11-30 17:21:42.20 5000000	98.93	
1.10	NORMAL	98.90	Raspberry-Pi	28.68	remaining_capacity_percentage	2024-11-30 17:19:54.98 2000000	98.9	
1.17	NORMAL	98.83	Raspberry-Pi	28.66	remaining_capacity_percentage	2024-11-30 17:18:58.35 4000000	98.83	

AWS IoT Rule Details:

- Description:** -
- ARN:** arn:aws:iot:region:account:rule/RouteToProcessTrashDataLambda
- Topic:** sahithiSensorTopic
- Status:** Active
- Created date:** November 23, 2024, 15:36:53 (UTC-08:00)

SQL statement:

```
SELECT * FROM `sahithiSensorTopic`
```

Actions:

Action	Service	Action
Lambda		Send a message to a Lambda function

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.

Bin bin1 - SEV3 - MEDIUM Capacity Alert

trash-data <no-reply@sns.amazonaws.com>
Yesterday at 5:28 PM
To: [REDACTED]

Bin Capacity Alert 🚨
Bin ID: bin1
Current Distance: 6.25 cm
Remaining Capacity: 21.55%
Filled Capacity: 78.45%
Severity: SEV3 - MEDIUM

Bin bin1 - SEV2 - HIGH Capacity Alert

trash-data <no-reply@sns.amazonaws.com>
Yesterday at 5:30 PM
To: [REDACTED]

Bin Capacity Alert 🚨
Bin ID: bin1
Current Distance: 3.95 cm
Remaining Capacity: 13.62%
Filled Capacity: 86.38%
Severity: SEV2 - HIGH

Bin bin1 - SEV3 - MEDIUM Capacity Alert

trash-data <no-reply@sns.amazonaws.com> Yesterday at 5:32 PM
To: [REDACTED]

🔴 Bin Capacity Alert 🔴
Bin ID: bin1
Current Distance: 8.22 cm
Remaining Capacity: 28.34%
Filled Capacity: 71.66%
Severity: SEV3 - MEDIUM

Bin bin1 - SEV2 - HIGH Capacity Alert

trash-data <no-reply@sns.amazonaws.com> Yesterday at 5:34 PM
To: [REDACTED]

🔴 Bin Capacity Alert 🔴
Bin ID: bin1
Current Distance: 2.96 cm
Remaining Capacity: 10.21%
Filled Capacity: 89.79%
Severity: SEV2 - HIGH

Bin bin1 - SEV2 - HIGH Capacity Alert

trash-data <no-reply@sns.amazonaws.com> Yesterday at 5:36 PM
To: [REDACTED]

🔴 Bin Capacity Alert 🔴
Bin ID: bin1
Current Distance: 3.29 cm
Remaining Capacity: 11.34%
Filled Capacity: 88.66%
Severity: SEV2 - HIGH

Bin bin1 - SEV1 - CRITICAL Capacity Alert

trash-data <no-reply@sns.amazonaws.com> Yesterday at 5:38 PM
To: [REDACTED]

🔴 Bin Capacity Alert 🔴
Bin ID: bin1
Current Distance: 2.63 cm
Remaining Capacity: 9.07%
Filled Capacity: 90.93%
Severity: SEV1 - CRITICAL

Bin bin1 - SEV2 - HIGH Capacity Alert

trash-data <no-reply@sns.amazonaws.com> Yesterday at 5:41 PM
To: [REDACTED]

🔴 Bin Capacity Alert 🔴
Bin ID: bin1
Current Distance: 2.96 cm
Remaining Capacity: 10.21%
Filled Capacity: 89.79%
Severity: SEV2 - HIGH

Bin bin1 - SEV2 - HIGH Capacity Alert

trash-data <no-reply@sns.amazonaws.com> Yesterday at 5:44 PM
To: [REDACTED]

🔴 Bin Capacity Alert 🔴
Bin ID: bin1
Current Distance: 5.61 cm
Remaining Capacity: 19.34%
Filled Capacity: 80.66%
Severity: SEV2 - HIGH

trash-data
Bin bin1 - SEV2 - HIGH Capacity Alert Yesterday
🔴 Bin Capacity Alert 🔴 Bin ID: bin1 Current Distance: 2.96 cm Remaining Capacity: 10.21% Filled Capacity: 89.79% Severity: SEV2 - HIGH

trash-data
Bin bin1 - SEV2 - HIGH Capacity Alert Yesterday
🔴 Bin Capacity Alert 🔴 Bin ID: bin1 Current Distance: 2.96 cm Remaining Capacity: 10.21% Filled Capacity: 89.79% Severity: SEV2 - HIGH

trash-data
Bin bin1 - SEV1 - CRITICAL Capacity Alert Yesterday
🔴 Bin Capacity Alert 🔴 Bin ID: bin1 Current Distance: 2.63 cm Remaining Capacity: 9.07% Filled Capacity: 90.93% Severity: SEV1 - CRITICAL

trash-data
Bin bin1 - SEV2 - HIGH Capacity Alert Yesterday
🔴 Bin Capacity Alert 🔴 Bin ID: bin1 Current Distance: 2.96 cm Remaining Capacity: 10.21% Filled Capacity: 89.79% Severity: SEV2 - HIGH

trash-data
Bin bin1 - SEV2 - HIGH Capacity Alert Yesterday
🔴 Bin Capacity Alert 🔴 Bin ID: bin1 Current Distance: 2.96 cm Remaining Capacity: 10.21% Filled Capacity: 89.79% Severity: SEV2 - HIGH

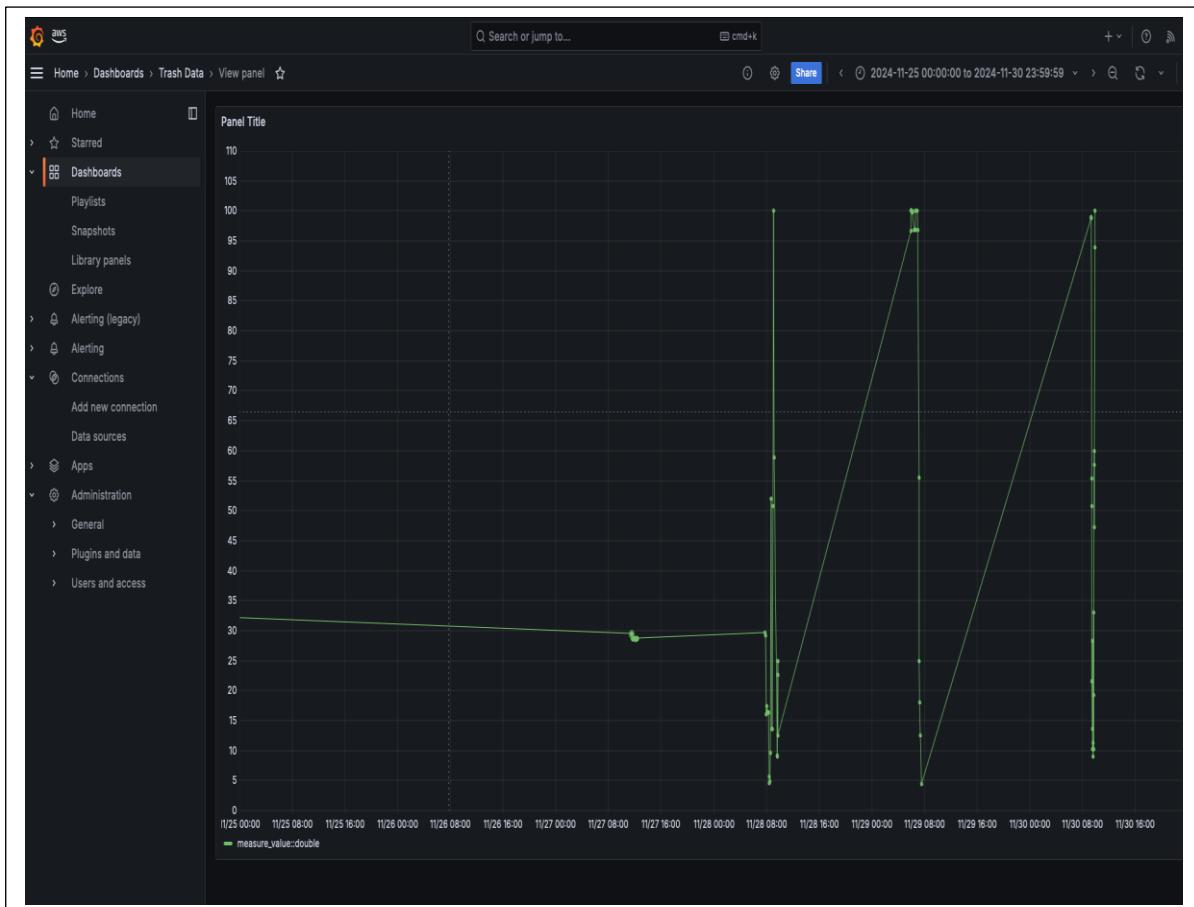
trash-data
Bin bin1 - SEV3 - MEDIUM Capacity Alert Yesterday
🔴 Bin Capacity Alert 🔴 Bin ID: bin1 Current Distance: 8.22 cm Remaining Capacity: 28.34% Filled Capacity: 71.66% Severity: SEV3 - MEDIUM

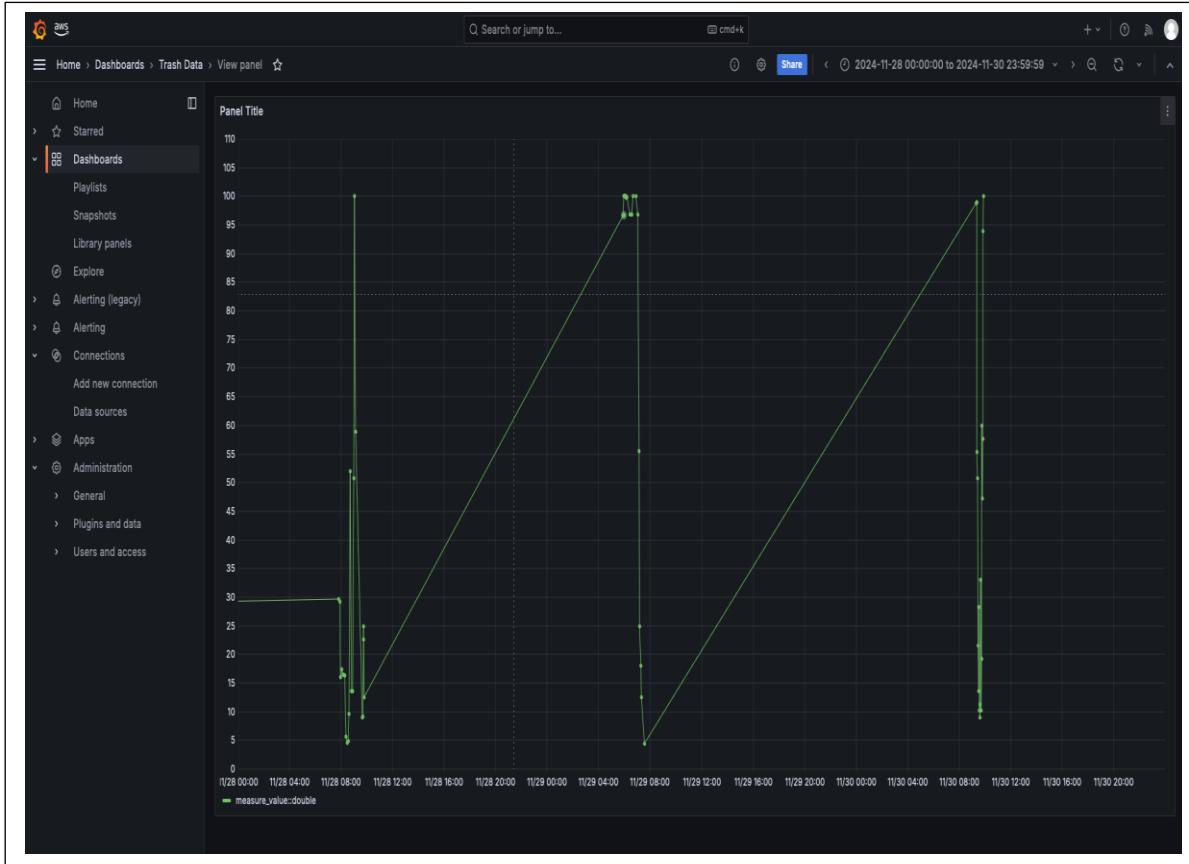
trash-data
Bin bin1 - SEV2 - HIGH Capacity Alert Yesterday
🔴 Bin Capacity Alert 🔴 Bin ID: bin1 Current Distance: 2.96 cm Remaining Capacity: 10.21% Filled Capacity: 89.79% Severity: SEV2 - HIGH

trash-data
Bin bin1 - SEV3 - MEDIUM Capacity Alert Yesterday
🔴 Bin Capacity Alert 🔴 Bin ID: bin1 Current Distance: 8.22 cm Remaining Capacity: 28.34% Filled Capacity: 71.66% Severity: SEV3 - MEDIUM

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.





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. Grafana Integration:** During the setup of the Grafana dashboard, I encountered permission restrictions while attempting to add Amazon Timestream as the data source. The workspace lacked the default Plugin Management feature. After identifying this issue, I enabled Plugin Management within the workspace settings, which allowed the integration to proceed.
- 4. Additionally, the timestamps in Grafana were initially displayed in UTC.** To address this, I adjusted the settings to display the time in PST, ensuring that the data aligned with local time zones.

5. AWS SNS Alert Configuration: I initially planned to use both email and SMS notifications for alerts. However, due to predefined limitations on the number of SMS messages allowed by AWS, I was unable to increase the quota. Despite reaching out to AWS customer service, I received no response. After consulting with my professor, I decided to proceed with email notifications exclusively for the alerting mechanism.

6. 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.

7. Debugging Lambda Functions: AWS Lambda functions initially failed to process data correctly due to improperly defined event triggers. Debugging these functions involved analysing CloudWatch logs.

Time spent to work on the project:

I spent nearly ~ 160 hours on this project for the past 1 month.