

## CSS532(Final Project - cloud Code and Configuration steps)

Name – Sahithi Chimakurthi

Student ID – 2303017

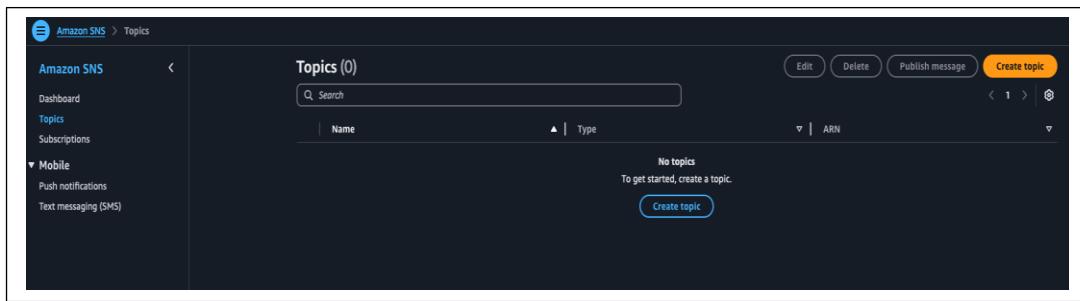
### 1. Setting up AWS IOT core: (already done during HW1 for CSS532)

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

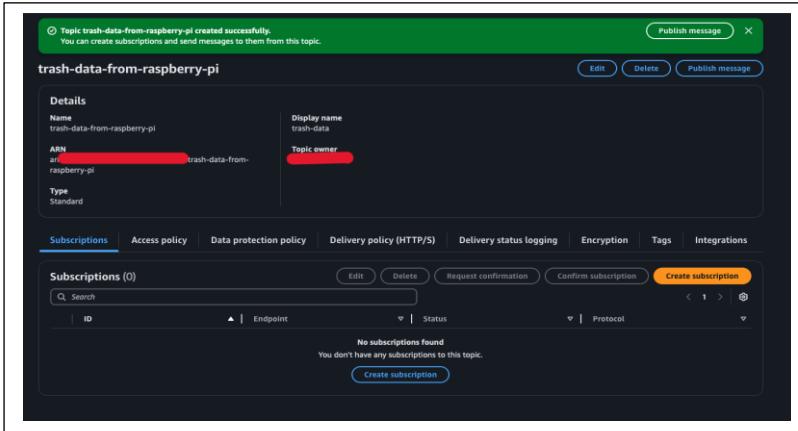
I created a new policy specifically for subscribing to a topic named “mySensorTopic”. Using the same Python script for HW1, I successfully subscribed to the topic by executing the script through the terminal. After configuring the policy, I used the AWS IoT console to subscribe to the same topic and 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.

### 2. Setting Up Amazon SNS (Simple Notification Service):

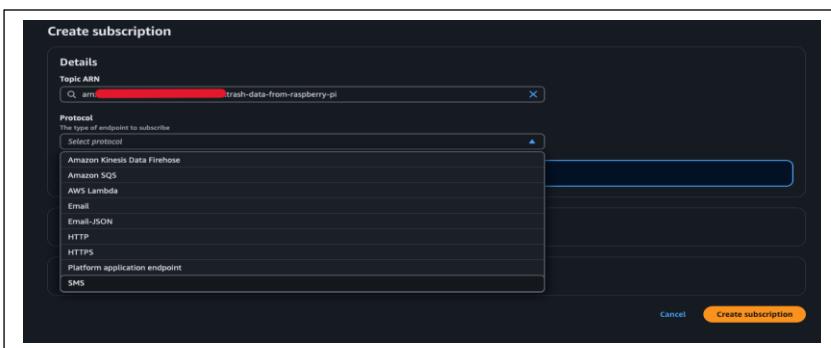
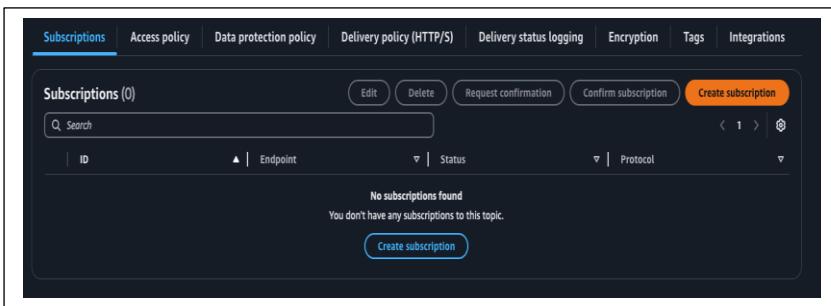
I navigated to the SNS Console by opening the AWS Management Console and accessing the SNS service. Once there, I created a topic by selecting "Create Topic" and filling in the necessary details, such as the topic name and type (e.g., Standard).

A screenshot of the 'Create topic' dialog box. It has a 'Details' section where the 'Type' is set to 'Standard'. There are two options: 'FIFO (first-in, first-out)' and 'Standard'. The 'Standard' option is selected and highlighted in blue. Below the type selection, there are fields for 'Name' (containing 'trash-data-from-raspberry-pi') and 'Display name - optional' (containing 'trash-data'). Both fields have character limits of 255 and 100 respectively. At the bottom of the dialog, there are 'Create topic' and 'Cancel' buttons.

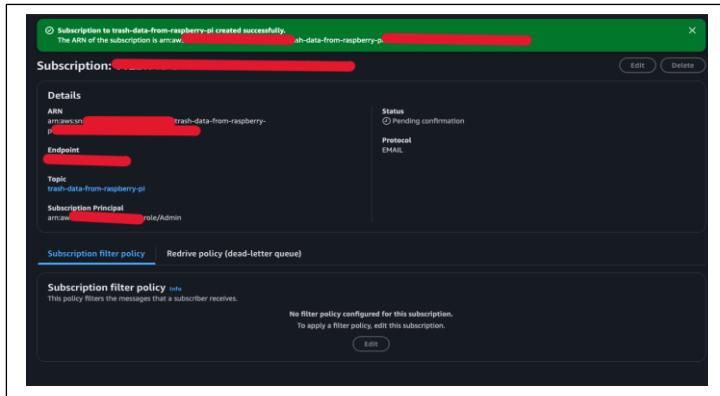
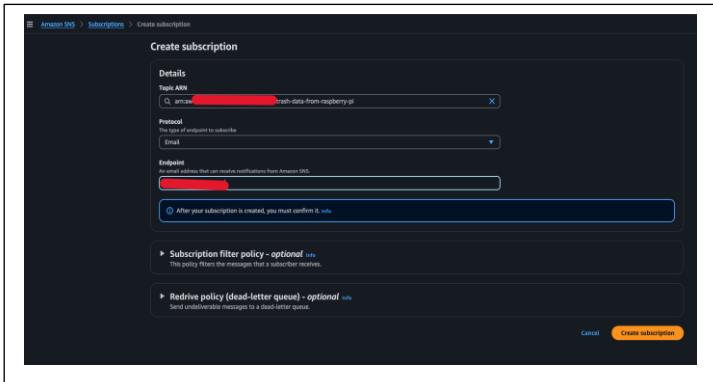
I assigned a unique name to the topic, such as trash-data-from-raspberry-pi, which reflects its purpose, and completed the topic creation process. The newly created topic was displayed in the list of available topics.



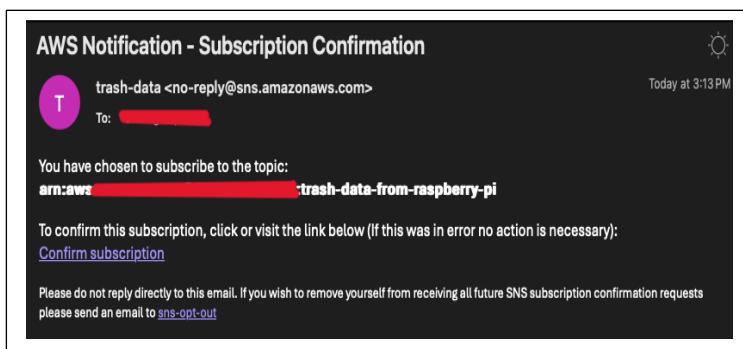
Next, I went to the Subscriptions tab under the created topic to set up a new subscription.



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.



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.



### 3. 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.

This screenshot shows the AWS Lambda Functions list page. At the top, there's a search bar labeled 'Filter by tags and attributes or search by keyword'. Below it, there are filters for 'Function name', 'Description', 'Package type', and 'Runtime'. On the right, there are buttons for 'Actions' and 'Create function'. The main area lists four functions: 'process-trash-data-from-raspberry-pi' (selected), 'process-trash-data-from-raspberry-pi-1', 'process-trash-data-from-raspberry-pi-2', and 'process-trash-data-from-raspberry-pi-3'. Each entry includes a preview icon, the function name, and a 'Last modified' timestamp.

This screenshot shows the 'Create function' wizard. It starts with a choice between 'Author from scratch' (selected), 'Use a blueprint' (with a sample Hello World example), and 'Container image'. The 'Basic information' step shows the function name 'process-trash-data-from-raspberry-pi' and runtime 'Python 3.13'. The 'Architecture' section shows 'x86\_64' selected. The 'Permissions' section indicates Lambda will create an execution role for CloudWatch Logs. There are also sections for 'Additional Configurations' and a 'Create function' button at the bottom.

This screenshot shows the 'Function overview' page for the 'process-trash-data-from-raspberry-pi' function. It displays the function name, a diagram showing a single function node, and a 'Layers' section with '(0)'. On the right, there are buttons for 'Throttle', 'Copy ARN', and 'Actions'. Below the diagram, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Configuration' tab is currently active.

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.

This screenshot shows the 'Configuration' tab in the Lambda console. The left sidebar lists various configuration options like General configuration, Triggers, Permissions (selected), Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, Asynchronous invocation, Code signing, File systems, and State machines. The 'Permissions' section shows the 'Execution role' set to 'process-trash-data-from-raspberry-pi-role'. The 'Resource summary' section shows a policy statement allowing access to Amazon CloudWatch Logs. The 'Lambda obtained this information from the following policy statements:' section lists two managed policies: 'AWSLambdaBasicExecutionRole' and 'AWSLambdaVPCExecutionRole'. The 'Statement' column for the first policy shows two entries: 'statement 0' and 'statement 1'.

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 configuration for 'process-trash-data-from-raspberry-pi-role'. The 'Permissions' tab is active, displaying an inline policy named 'publish-to-sns'. The policy document is as follows:

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

The policy grants the sns:Publish action to the specified SNS topic resource. The ARN of the role is listed at the top: arn:aws:iam::region:role/service-role/process-trash-data-from-raspberry-pi-role.

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.

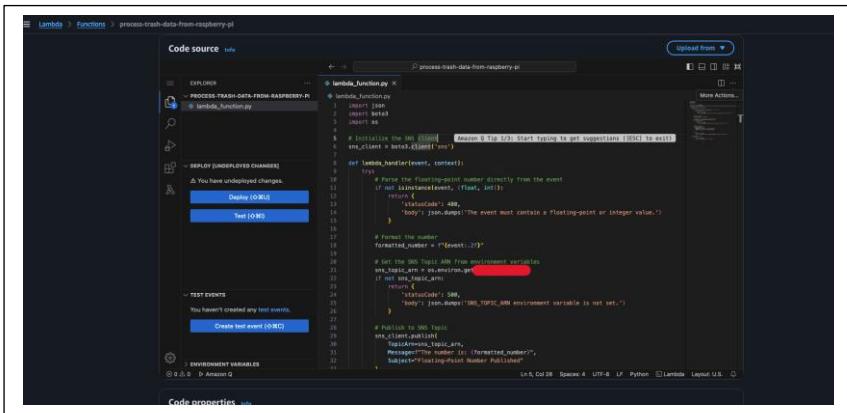
The screenshot shows the 'Specify permissions' dialog for creating a new inline policy. The JSON editor contains the following policy document:

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

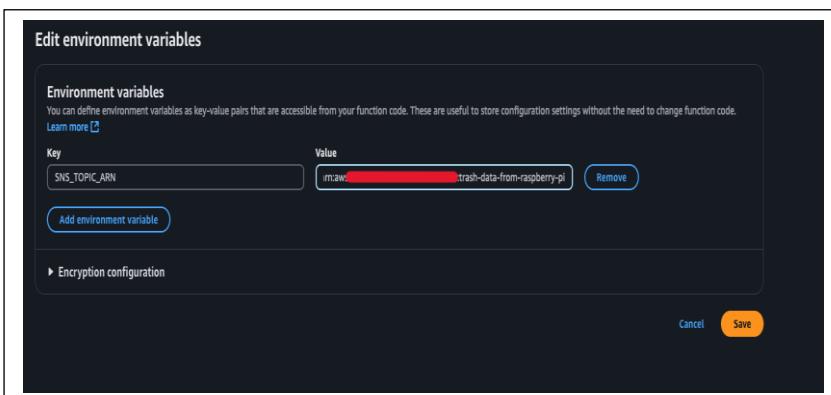
The right side of the dialog shows the service selection interface, where 'sns' is chosen from the available services. The sidebar also lists other services like API Gateway and CloudWatch Metrics.

The screenshot shows the IAM role configuration for 'process-trash-data-from-raspberry-pi-role' after the policy has been attached. The 'Permissions' tab is active, showing two policies: 'AWSLambdaBasicExecutionRole' (Customer managed) and 'publish-to-sns' (Customer inline). The 'publish-to-sns' policy is the one we just created, containing the sns:Publish permission. The ARN of the role is listed at the top: arn:aws:iam::region:role/service-role/process-trash-data-from-raspberry-pi-role.

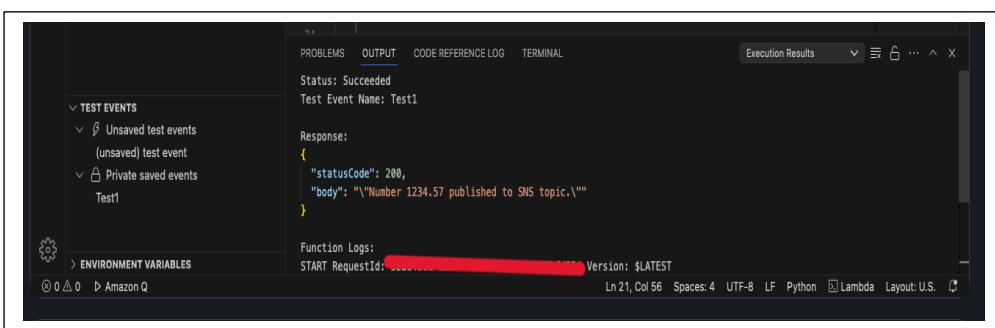
I then wrote the Lambda function's code, which included logic to publish alert messages to the SNS topic.



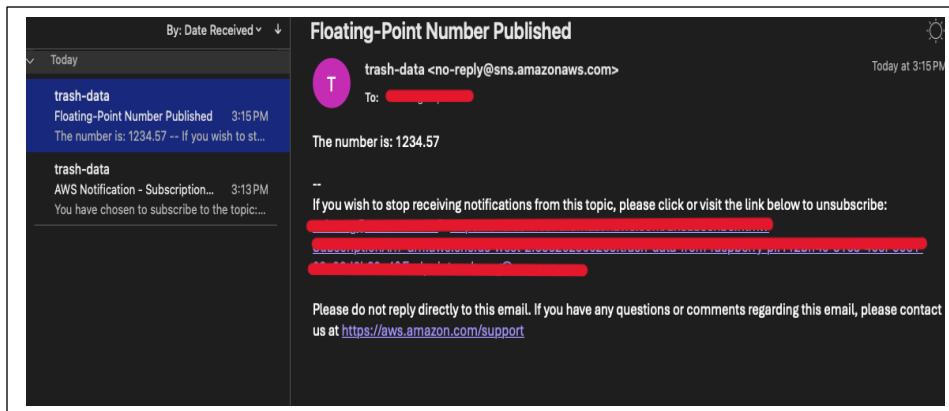
To make the function dynamic and maintainable, I used environment variables to store the SNS topic ARN.



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.



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

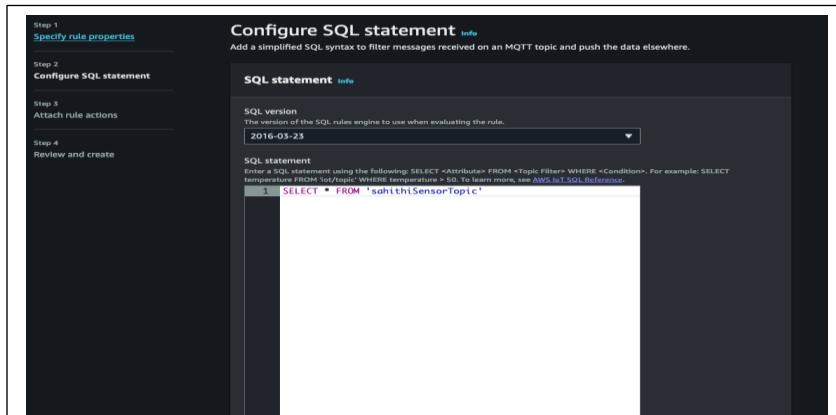


## 4. Setting Up AWS IoT Core Rules:

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”.



The screenshot shows the AWS IoT Rule creation process. In Step 2: SQL statement, the SQL query is set to `SELECT \* FROM 'sahithiSensorTopic'`. In Step 3: Rule actions, a Lambda function named `arn:aws:lambda:us-east-1:123456789012:function:process-trash-data-from-raspberry-pi` is selected as the action. The Lambda function version is set to \$LATEST. There is also an 'Error action' section with the note 'No error action'. At the bottom, there are 'Cancel', 'Previous', and 'Create' buttons.

Successfully created rule RouteToProcessTrashDataLambda.

AWS IoT > Message routing > Rules > RouteToProcessTrashDataLambda

**RouteToProcessTrashDataLambda** Info

**Details**

Description

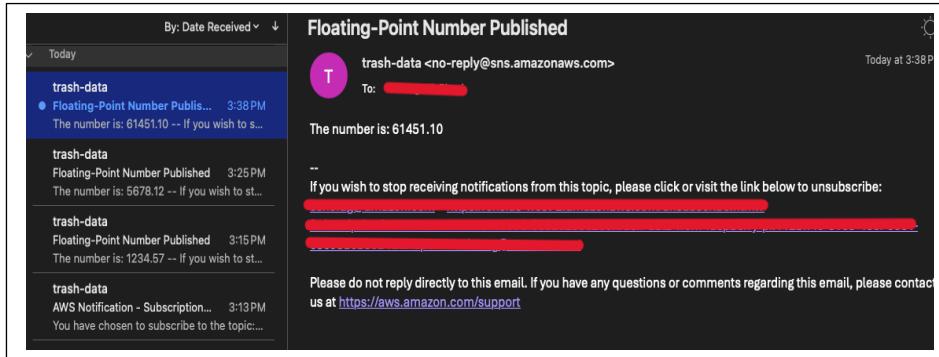
I created a new IoT rule, assigned it a descriptive name “RouteToProcessTrashDataLambda” and linked the Lambda function as the action to execute when the rule was triggered.

The screenshot shows the AWS Lambda function configuration for `process-trash-data-from-raspberry-pi`. It includes a function overview with a diagram showing triggers from AWS IoT and destinations to Amazon SNS. The configuration tab is selected, showing triggers (1), permissions, destinations, function URL, and environment variables. A trigger named "AWS IoT: RouteToProcessTrashDataLambda" is listed under triggers.

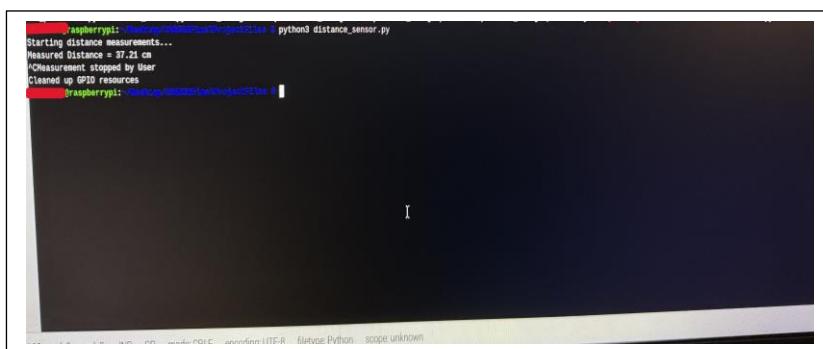
After successfully creating the IoT rule, I tested the setup by publishing test data to the Topic using a Python script.

```
python3 testFloatNumbers.py
Connecting to endpoint [REDACTED] with client ID 'sahithiClient'...
Connected!
Subscribing to topic 'sahithiSensorTopic'...
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!
```

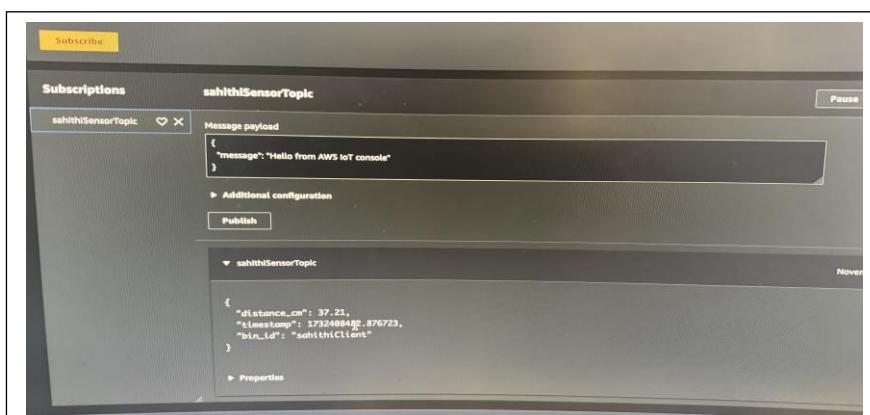
I verified that the IoT Core rule routed the data to the Lambda function and triggered an SNS notification. Additionally, I validated that all components, including email notifications, worked as intended by monitoring their responses in real time.



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

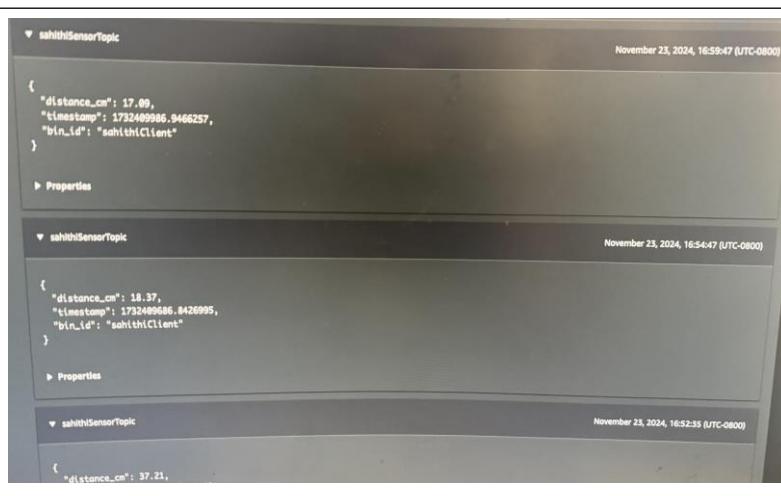


I confirmed that the data was received in the IoT console under the sensor Topic, and appropriate alerts were triggered.



```
sahithic@raspberrypi:~/Desktop/CSS532FinalProjectFiles $ python3 distance_sensor.py
Starting distance measurements...
Measured Distance = 18.37 cm
Measured Distance = 17.69 cm
Measured Distance = 3.95 cm
Measurement stopped by User
Cleaned up GPIO resources
sahithic@raspberrypi:~/Desktop/CSS532FinalProjectFiles $
```

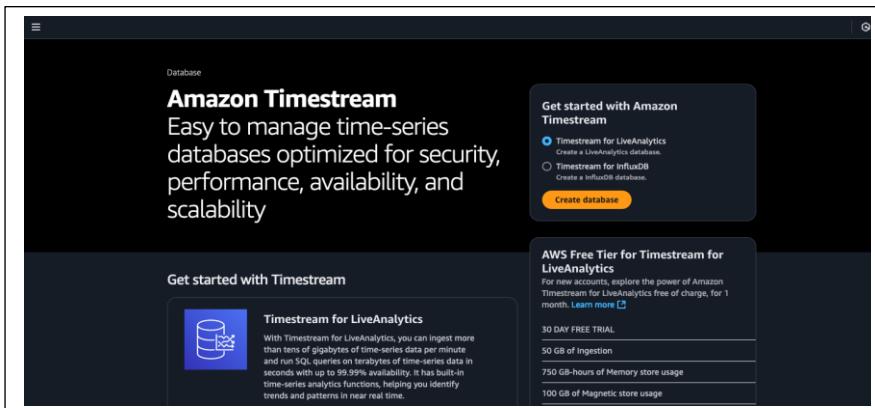
```
(aws-iot-env) sahithic@raspberrypi:~/Desktop/CSS532FinalProjectFiles $ python3 aws_publisher.py
Connecting to AWS IoT Core at [REDACTED]
Connected to AWS IoT Core!
Starting AWS IoT publisher...
Published measurement: {'distance_cm': 37.21, 'timestamp': 1732468482.876723, 'bin_id': 'sahithiClient'}
Published measurement: {'distance_cm': 18.37, 'timestamp': 1732469986.842695, 'bin_id': 'sahithiClient'}
Published measurement: {'distance_cm': 17.69, 'timestamp': 1732469986.9466257, 'bin_id': 'sahithiClient'}
```



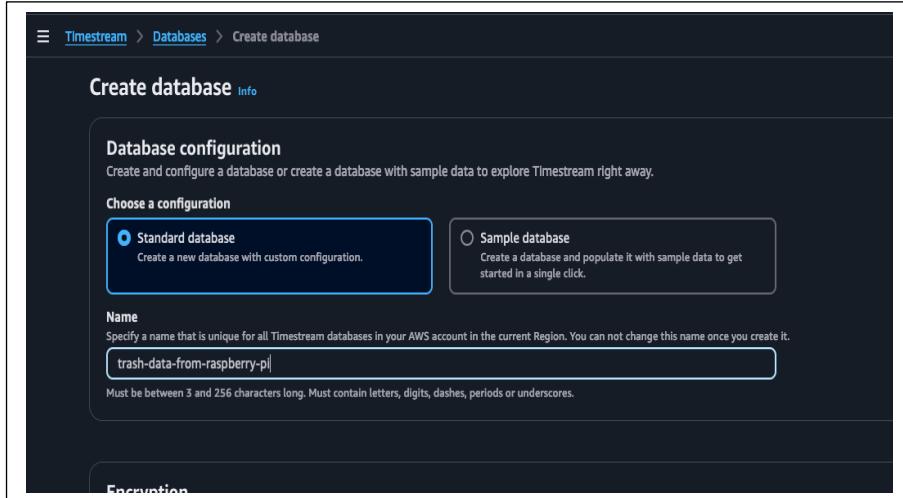
All notifications were successfully delivered to the email subscribers, ensuring seamless integration and functionality across the system.

## 5. Setting up TimeStream database:

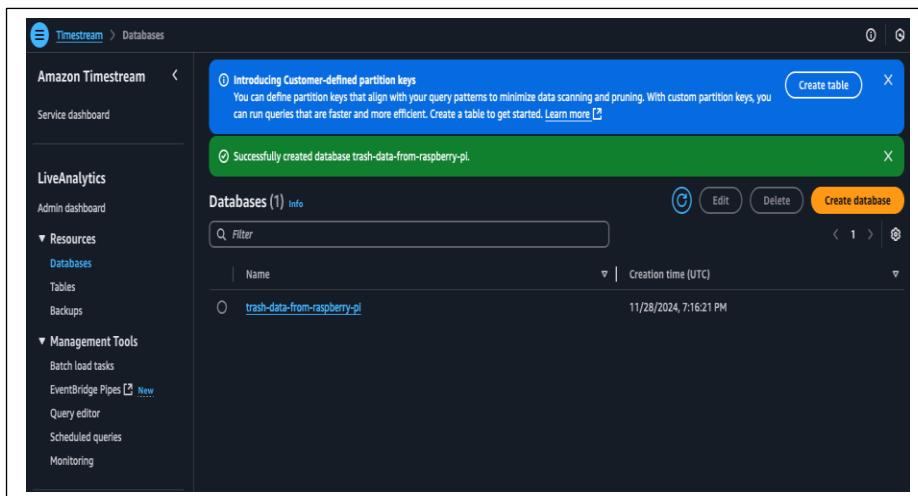
To set up and utilize Amazon Timestream for real-time analytics, I began by navigating to the Timestream Console on the AWS Management Console.



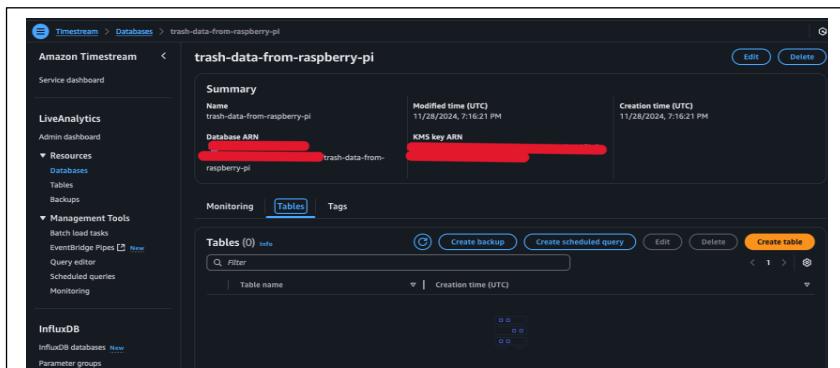
I selected the option to create a new database, opting for the "Standard database" type to suit the project's needs. I named it as "trash-data-from-raspberry-pi".



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.



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.

The screenshot shows the Amazon Timestream console under the 'Databases' section. A database named 'trash-data-from-raspberry-pi' is selected. The 'Summary' tab is active, displaying details like Name (trash-data-from-raspberry-pi), Modified time (UTC) (11/28/2024, 7:16:21 PM), and Creation time (UTC) (11/28/2024, 7:16:21 PM). The 'Tables' tab is selected, showing a table count of 0. There are buttons for 'Create backup', 'Create scheduled query', 'Edit', 'Delete', and 'Create table'. The sidebar on the left includes sections for Service dashboard, LiveAnalytics (Admin dashboard, Resources, Databases, Tables, Backups), Management Tools (Batch load tasks, EventBridge Pipes, Query editor, Scheduled queries, Monitoring), and InfluxDB (InfluxDB databases, Parameter groups).

The screenshot shows the 'Create table' configuration dialog. It includes sections for Data retention info, Memory store retention (12 hours), Magnetic store retention (10 years), Magnetic Storage Writes (checkbox for Enable magnetic storage writes), and Backup settings (checkbox for Turn on backups for this table). The 'Data retention info' section notes that data moves from memory to magnetic store as it ages, and that data exceeding magnetic store retention will be deleted.

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

The screenshot shows the Amazon Timestream console under the 'Tables' section. A table named 'distance-data' is listed, which was successfully created. The table has 1 row, belongs to the 'trash-data-from-raspberry-pi' database, and was created on 11/28/2024, 7:24:48 PM. There are buttons for 'Create backup', 'Create scheduled query', 'Edit', 'Delete', and 'Create table'. The sidebar on the left is identical to the previous screenshot.

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.

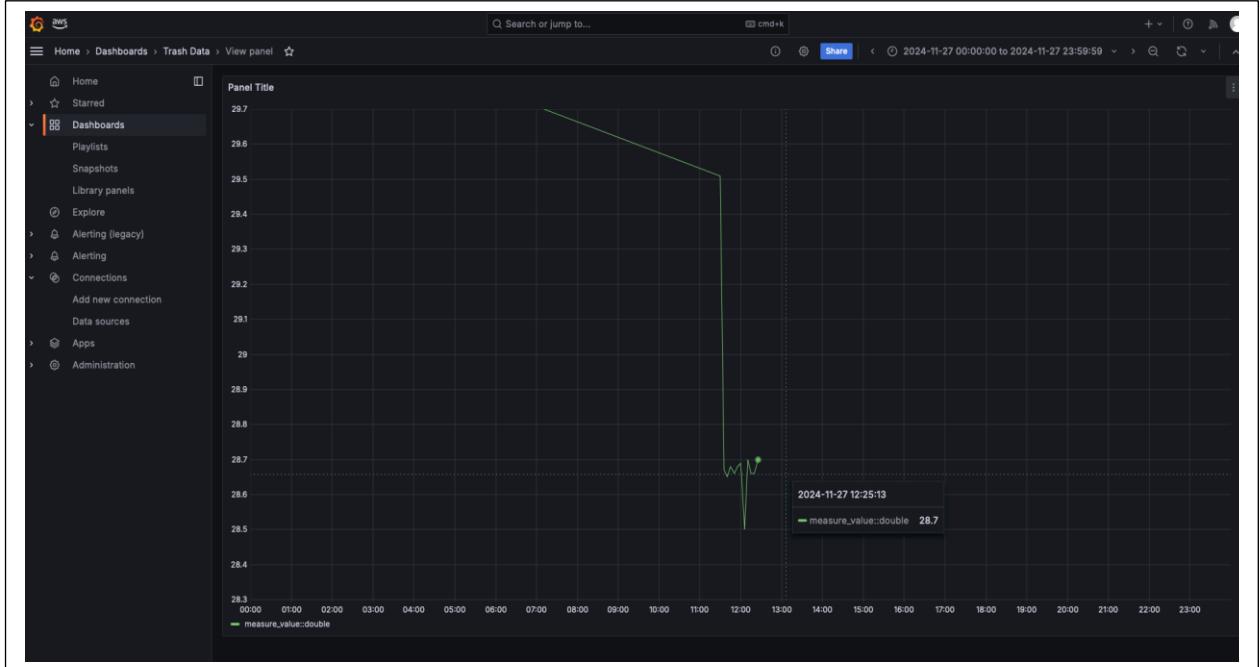
The screenshot shows the AWS Timestream Query Editor interface. The 'Tables (1)' section lists a single table named 'distance-data'. The main area displays a SQL query: 'SELECT \* FROM "trash-data-from-raspberry-pi"."distance-data"'. Below the query, the status bar indicates 'Success' and '0 rows affected.' A red box highlights the status bar.

This screenshot is similar to the first one, but the status bar now shows '24 rows returned.' A red box highlights the status bar.

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_na me	time	measure_value::double
bin1	Raspberry-PI	trash_bin_re maining_ca pacity	2024-11-27 20:25:13.10 6000000	28.7
bin1	Raspberry-PI	trash_bin_re maining_ca pacity	2024-11-27 20:20:13.00 2000000	28.66
bin1	Raspberry-PI	trash_bin_re maining_ca pacity	2024-11-27 20:15:12.89 7000000	28.66
bin1	Raspberry-PI	trash_bin_re maining_ca pacity	2024-11-27 20:10:12.79 3000000	28.7
bin1	Raspberry-PI	trash_bin_re maining_ca pacity	2024-11-27 20:05:12.68 9000000	28.5
bin1	Raspberry-PI	trash_bin_re maining_ca pacity	2024-11-27 20:00:12.58 4000000	28.69
bin1	Raspberry-PI	trash_bin_re maining_ca pacity	2024-11-27 19:55:12.48 0000000	28.68
bin1	Raspberry-PI	trash_bin_re maining_ca pacity	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.

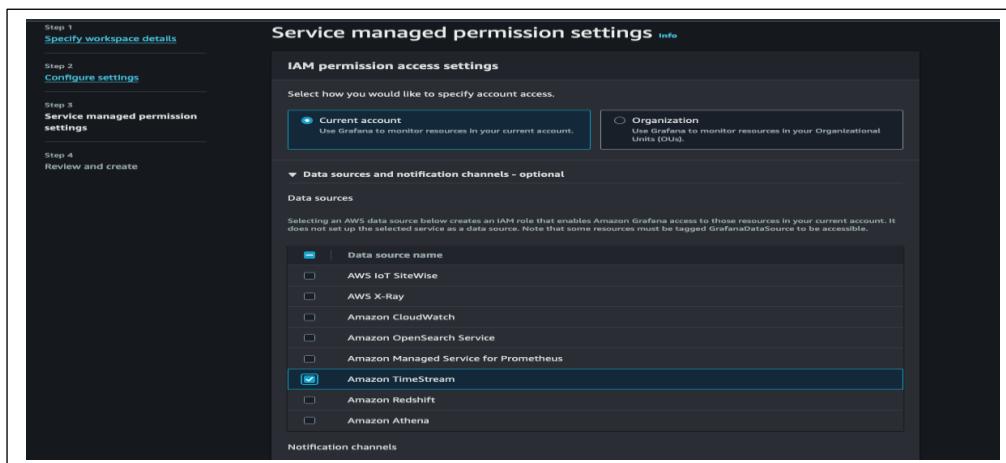
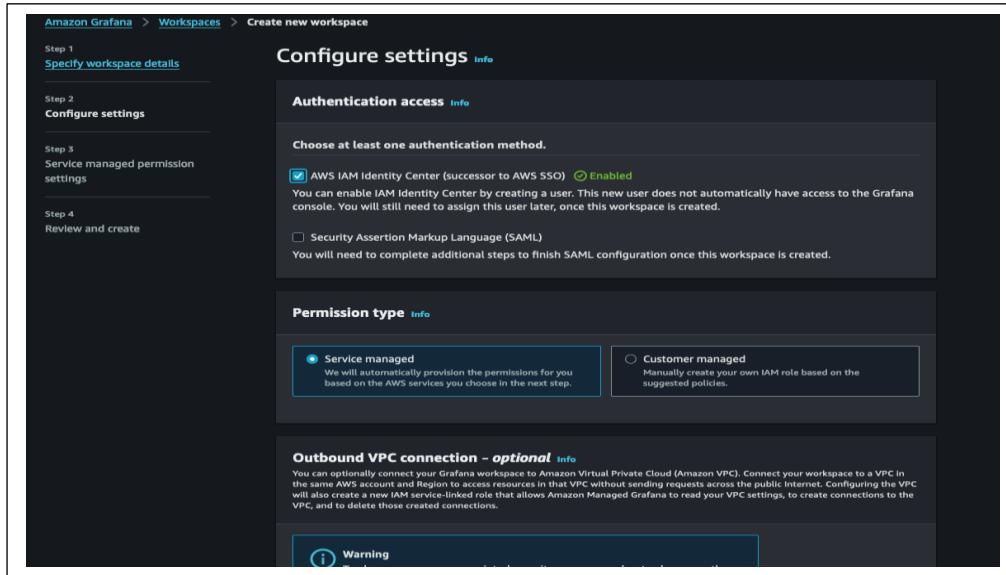


## 6. Setting up Graphana dashboard:

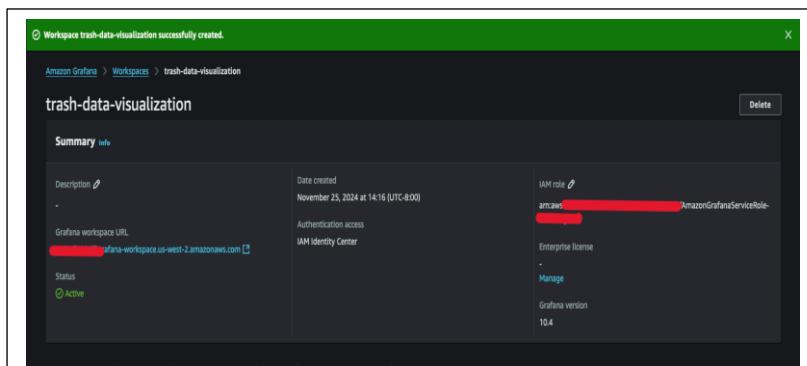
To set up and utilize Grafana for visualizing data stored in Amazon Timestream, I started by creating a new Grafana workspace in the AWS Management Console.

A screenshot of the "Create new workspace" wizard in the Amazon Managed Grafana console. The wizard is at Step 1: "Specify workspace details". The workspace name is set to "trash-data-visualization". The workspace description is optional. The Grafana version is set to 10.4. There is a section for "Tags - optional" where users can add new tags, with a note that they can add up to 50 more tags. The "Next" button is visible at the bottom right.

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.



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



Initially, no users were added to the workspace.

The screenshot shows the 'AWS IAM Identity Center (successor to AWS SSO)' configuration page. At the top, there are tabs for Authentication, Data sources, Notification channels, Tags, Network access control, and Workspace configuration options - new. Below these tabs, a section titled 'AWS IAM Identity Center (successor to AWS SSO)' contains a note: 'You can enable AWS IAM Identity Center by creating a user or connect IAM Identity Center to an external identity provider (IdP) to enable users to log in to the workspace with their existing credentials. Note that when you enable IAM Identity Center by creating a new user, you will need to assign this user access to the workspace before they can log in to the workspace.' A button labeled 'Assign new user or group' is present, along with a note: '⚠ 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 first screenshot shows the 'Assign user' step in the AWS IAM Identity Center. It lists one user under 'Selected users and groups (1)'. The second screenshot shows the 'Assigned users' list, which also contains one user. Both screenshots include 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.

The screenshot shows the 'Assigned users' list in the AWS IAM Identity Center. A context menu is open over a selected user, with the 'Action' dropdown menu showing options: 'Assign user', 'Unassign user', 'Make admin' (which is highlighted), 'Make editor', and 'Make viewer'. The user list shows one user assigned as a 'Viewer'.

The screenshot shows the AWS IAM Identity Center interface. The top navigation bar includes 'Amazon Grafana > Workspaces > trash-data-visualization > AWS IAM Identity Center (successor to AWS SSO)'. Below the navigation is a header 'AWS IAM Identity Center (successor to AWS SSO)' with a 'Delete configuration' button. A tabs section has 'Assigned users' selected, with 'Assigned user groups' as an option. Under 'Users (1) Info', it says 'The following users have already been assigned access to Grafana.' A search bar 'Find users' is present. A table lists one user: 'Full name' (redacted), 'User type' (Admin), and an 'Action' dropdown menu. Navigation icons like back, forward, and search are at the bottom.

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 image contains four screenshots:

- Top Left:** 'trash-data-visualization' workspace summary page. It shows a 'Description' field with a redacted note, a 'Grafana workspace URL' field containing 'grafana-workspace.us-west-2.amazonaws.com' with a copy icon, and a 'Status' field showing 'Active' with a green checkmark.
- Top Right:** 'Welcome to Amazon Managed Grafana' screen. It features the AWS logo and a 'Sign in with AWS IAM Identity Center' button. Below the button are links for 'Documentation', 'Support', 'Community', 'Pro (Licensed)', 'Cookies', and 'Amazon Grafana (Licensed) | v10.4.1'.
- Bottom Left:** 'Sign in' screen for Amazon Managed Grafana. It has fields for 'Username' (redacted) and 'Password' (redacted), with 'Show password' and 'Forgot password' links, and 'Sign In' and 'Cancel' buttons. A blue 3D cube icon is to the left of the form.
- Bottom Right:** 'Welcome to Amazon Managed Grafana' dashboard. It displays sections for 'Get started', 'Data source', 'Dashboards', and 'Learn how to use Grafana'. It also includes a sidebar with navigation links like Home, Starred, Dashboards, Explore, Alerting, Connections, Data sources, Apps, and Administration.

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

The screenshot shows the 'Add new connection' interface in Grafana. The left sidebar has 'Home', 'Starred', 'Dashboards', 'Explore', 'Alerting (legacy)', 'Alerting', 'Connections', 'Add new connection' (which is selected and highlighted in orange), 'Data sources', 'Apps', and 'Administration'. The main area is titled 'Add new connection' with a sub-section 'Browse and create new connections'. A search bar contains 'timestream'. Below it is a 'Data sources' section with a card for 'Amazon Timestream' featuring its logo.

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

The screenshot shows the 'Add new connection' interface for Amazon Timestream. The left sidebar has 'Data sources' selected. The main content area displays the 'Amazon Timestream' plugin details, including its description as a 'Managed timeseries database from amazon' and a note that the user does not have permission to install it. It includes sections for 'Overview', 'Version history', 'Timestream Datasource', 'Add the data source' (with steps 1-3), 'Authentication' (link to AWS authentication topic), and 'IAM policies' (link to IAM permissions for Timestream API).

The screenshot shows the 'Workspace configuration options - new' tab. It contains two sections: 'Grafana alerting' (Info: Prometheus alert rules, status Off) and 'Plugin management - new' (Info: Turn plugin management on to allow workspace admins to discover, install, update, and uninstall plugins). Both sections have an 'Edit' button.

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

The screenshot shows the 'Amazon Grafana > Workspaces > trash-data-visualization > Workspace configuration options' page. It displays the 'Grafana alerting' and 'Plugin management - new' sections from the previous screenshot. In the 'Plugin management' section, the checkbox 'Turn plugin management on' is checked. At the bottom right are 'Cancel' and 'Save changes' buttons.

The screenshot shows the Amazon Grafana workspace summary page. At the top, a green banner displays the message "The workspace updated successfully." Below the banner, the page title is "trash-data-visualization". The "Summary" tab is selected. Key details shown include:

- Description: -
- Date created: November 25, 2024 at 14:16 (UTC-8:00)
- Grafana workspace URL: [REDACTED]afana-workspace.us-west-2.amazonaws.com [Copy]
- Authentication access: IAM Identity Center
- Status: Active

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 Grafana plugin store. The search bar is set to "Amazon Timestream". The results page for "Amazon Timestream" is displayed, showing the following details:

- Version: 2.9.11
- From: Grafana Labs
- Downloads: 8,364,684
- Dependencies: Grafana >=9.5.13
- Signature: Signed
- Install 2.9.11

The "Overview" tab is selected. The page includes sections for "Timestream Datasource" and "Add the data source". It also contains sections for "Authentication" and "IAM policies".

The screenshot shows the Grafana configuration screen under "Connections". A modal window titled "Installing latest version of Amazon Timestream" is open, displaying the message "This process may take a few minutes to complete." The status bar at the bottom of the modal indicates "1/694". The main configuration screen shows the "Amazon Timestream" plugin details, including its version (2.9.11), dependencies (Grafana >=9.5.13), and signature (Signed). Buttons for "Uninstall", "Update", and "Add new data source" are visible.

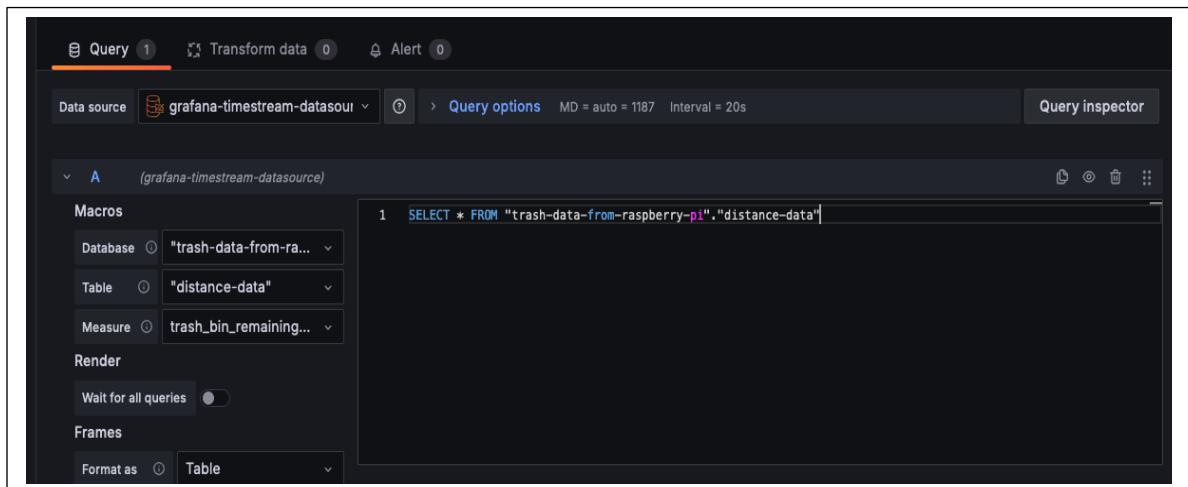
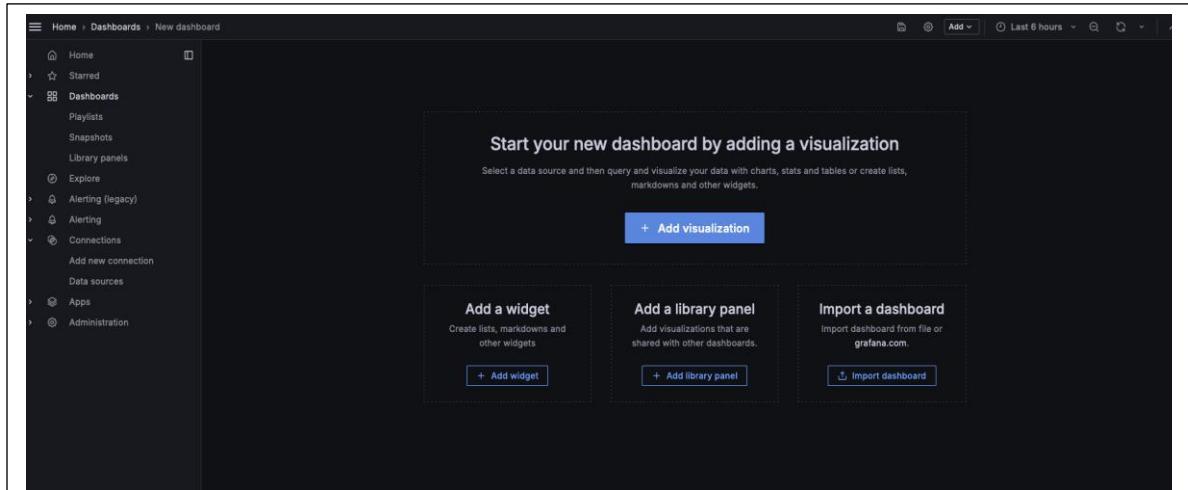
The screenshot shows the Grafana plugin store page for the "Amazon Timestream" plugin. At the top, there's a header with the plugin name, a "Website" link, and a "From Grafana Labs" badge. Below the header, there are tabs for "Overview" (which is selected) and "Version history". The main content area is titled "Timestream Datasource" and contains a brief description: "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". Under this, there's a section titled "Add the data source" with three steps: 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. There are also sections for "Authentication" (with a note about AWS authentication) and "IAM policies".

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. At the top, it says "Type: Amazon Timestream". Below that are tabs for "Settings" (which is selected), "Dashboards", "Permissions", and "Insights". The "Name" field is set to "grafana-timestream-datasource" and has a "Default" toggle switch turned on. The "Connection Details" section contains fields for "Authentication Provider", "Assume Role ARN", "External ID", "Endpoint", and "Default Region" (set to "us-west-2"). The "Timestream Details" section contains fields for "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 again, but now with a green success message at the top: "Connection success". The message continues: "Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view." Below the message are "Delete" and "Save & test" buttons.

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



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



A BinID	A Source	A measure_name	O time	measure_value::double
	Raspberry-Pi	trash_bin_remaining_capacity	2024-11-24 21:05:45	26431
	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.

Save dashboard

New dashboard

Details

Title:

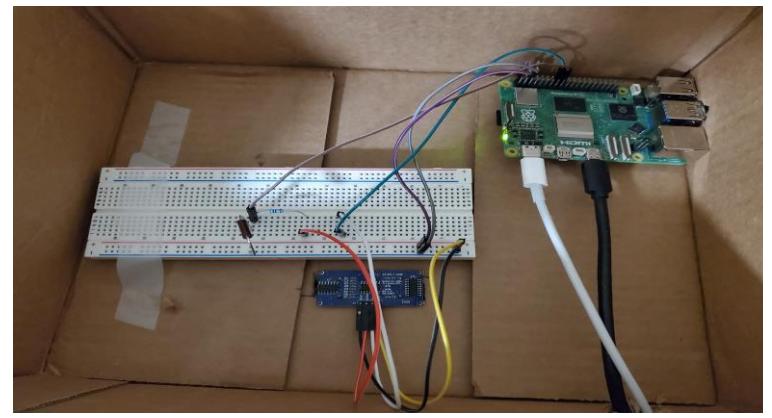
Description:

Folder:



## 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.

```
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 = 3.05 cm
Measurement saved successfully.
Measured Distance = 8.22 cm
Measurement saved successfully.
Measured Distance = 2.98 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 = 2.98 cm
Measurement saved successfully.
Measured Distance = 5.61 cm
Measurement saved successfully.
Measured Distance = 17.40 cm
Measurement saved successfully.
Measured Distance = 13.78 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': 1733016792.233702, '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'}
^C
Publisher stopped by user
Disconnected from AWS IoT Core
```

**Subscriptions**

**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, 17:41:45 (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

The screenshot shows the AWS IoT Core Publish interface. It displays two messages published to the topic 'sahithiSensorTopic'. The first message was published at November 30, 2024, 17:53:43 (UTC-0800) and contains the following JSON payload:

```
{
  "distance_cm": 29.54,
  "timestamp": 1733018022.2958603,
  "bin_id": "bin1"
}
```

The second message was published at November 30, 2024, 17:51:43 (UTC-0800) and contains the following JSON payload:

```
{
  "distance_cm": 27.23,
  "timestamp": 1733017902.2893374,
  "bin_id": "bin1"
}
```

**(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.

The screenshot shows the Amazon Timestream Query results page. The table has the following columns:

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 17:39:42.25 4000000	33.07	
90.93	SEV1 - CRITICAL	9.07	Raspberry-Pi	2.63	remaining_capacity_percentage	2024-11-30 17:37:42.24 8000000	9.07	
88.66	SEV2 - HIGH	11.34	Raspberry-Pi	3.29	remaining_capacity_percentage	2024-11-30 17:35:42.24 3000000	11.34	
89.79	SEV2 - HIGH	10.21	Raspberry-Pi	2.96	remaining_capacity_percentage	2024-11-30 17:33:42.23 8000000	10.21	
71.66	SEV3 - MEDIUM	28.34	Raspberry-Pi	8.22	remaining_capacity_percentage	2024-11-30 17:31:42.23 3000000	28.34	
86.58	SEV2 - HIGH	13.62	Raspberry-Pi	3.95	remaining_capacity_percentage	2024-11-30 17:29:42.22 8000000	13.62	
78.45	SEV3 - MEDIUM	21.55	Raspberry-Pi	6.25	remaining_capacity_percentage	2024-11-30 17:27:42.22 1000000	21.55	
49.21	NORMAL	50.79	Raspberry-Pi	14.73	remaining_capacity_percentage	2024-11-30 17:25:42.21 6000000	50.79	
44.59	NORMAL	55.41	Raspberry-Pi	16.07	remaining_capacity_percentage	2024-11-30 17:23:42.21 1000000	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

AWS IoT > Message routing > Rules > RouteToProcessTrashDataLambda

**RouteToProcessTrashDataLambda** Info

**Details**

Description:

ARN:  /RouteToProcessTrashDataLambda

Topic:  sahithiSensorTopic

Status:  Active

Created date: November 23, 2024, 15:36:33 (UTC-08:00)

**SQL statement**

SQL statement:  SELECT \* FROM 'sahithiSensorTopic'

SQL version: 2016-03-23

**Actions (1)**

Actions occur when an event is triggered. Actions are executed until all actions are completed or an error occurs. To add or remove actions, you will need to edit the rule.

Action	Service	Retain message	Send a message to a Lambda function
Lambda	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

View details

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookies

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>  
To: [REDACTED]  
Yesterday at 5:28 PM  
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>  
To: [REDACTED]  
Yesterday at 5:30 PM  
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>  
To: [REDACTED]  
Yesterday at 5:32 PM  
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>  
To: [REDACTED]  
Yesterday at 5:34 PM  
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>  
To: [REDACTED]  
Yesterday at 5:36 PM  
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>  
To: [REDACTED]  
Yesterday at 5:38 PM  
Bin Capacity Alert 🔞  
Bin ID: bin1  
Current Distance: 2.63 cm  
Remaining Capacity: 9.07%  
Filled Capacity: 90.93%  
Severity: SEV1 - CRITICAL

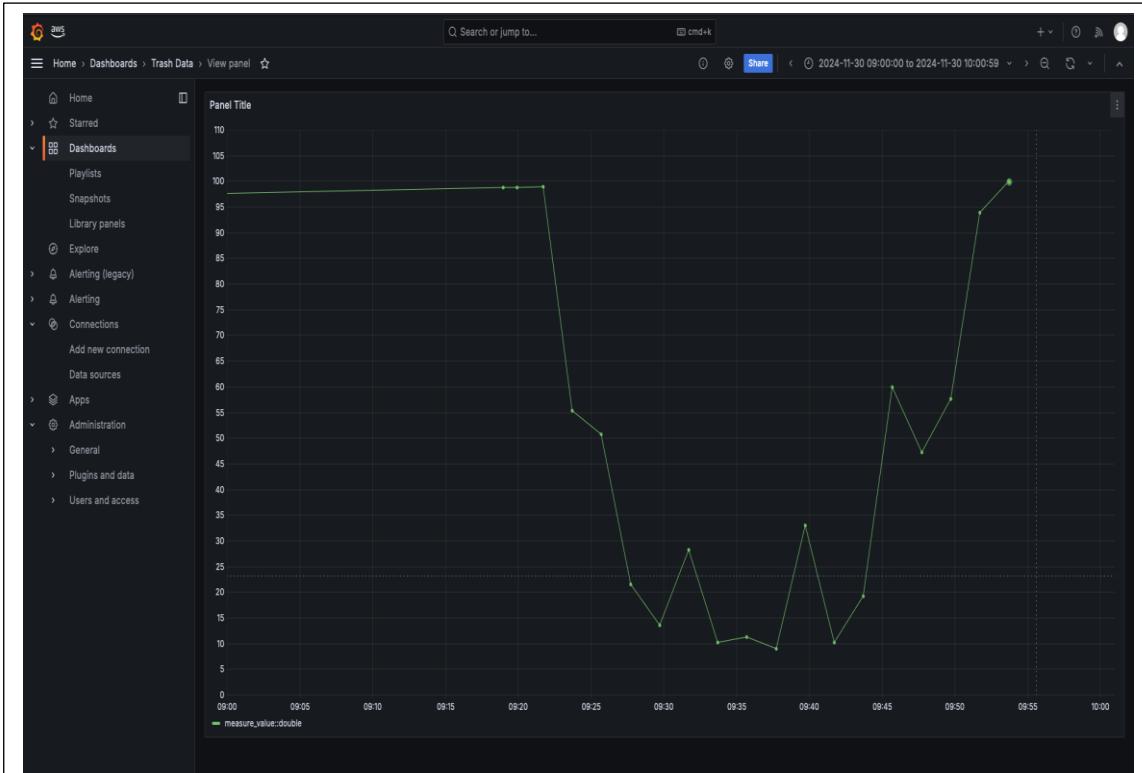
**Bin bin1 - SEV2 - HIGH Capacity Alert**  
trash-data <no-reply@sns.amazonaws.com>  
To: [REDACTED]  
Yesterday at 5:41 PM  
Bin Capacity Alert 🔞  
Bin ID: bin1  
Current Distance: 2.96 cm  
Remaining Capacity: 10.21%  
Filled Capacity: 89.79%  
Severity: SEV2 - HIGH

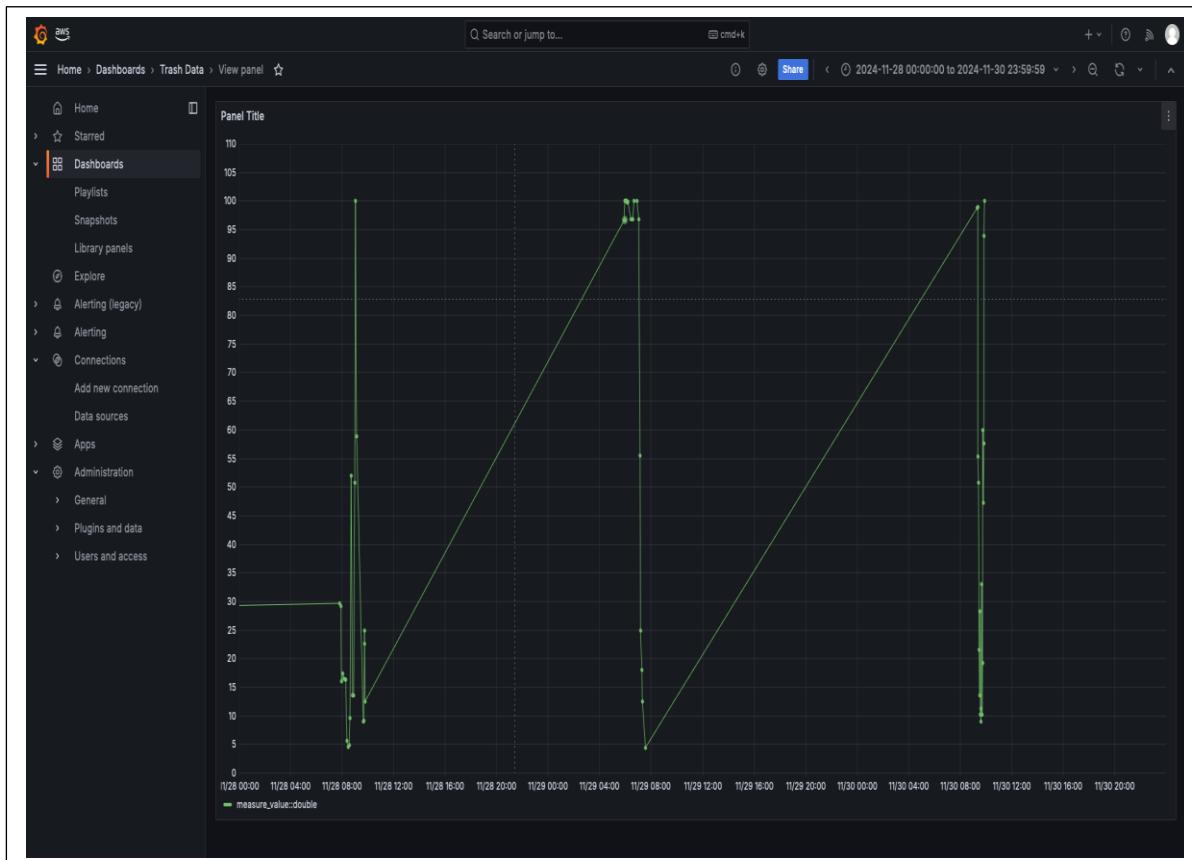
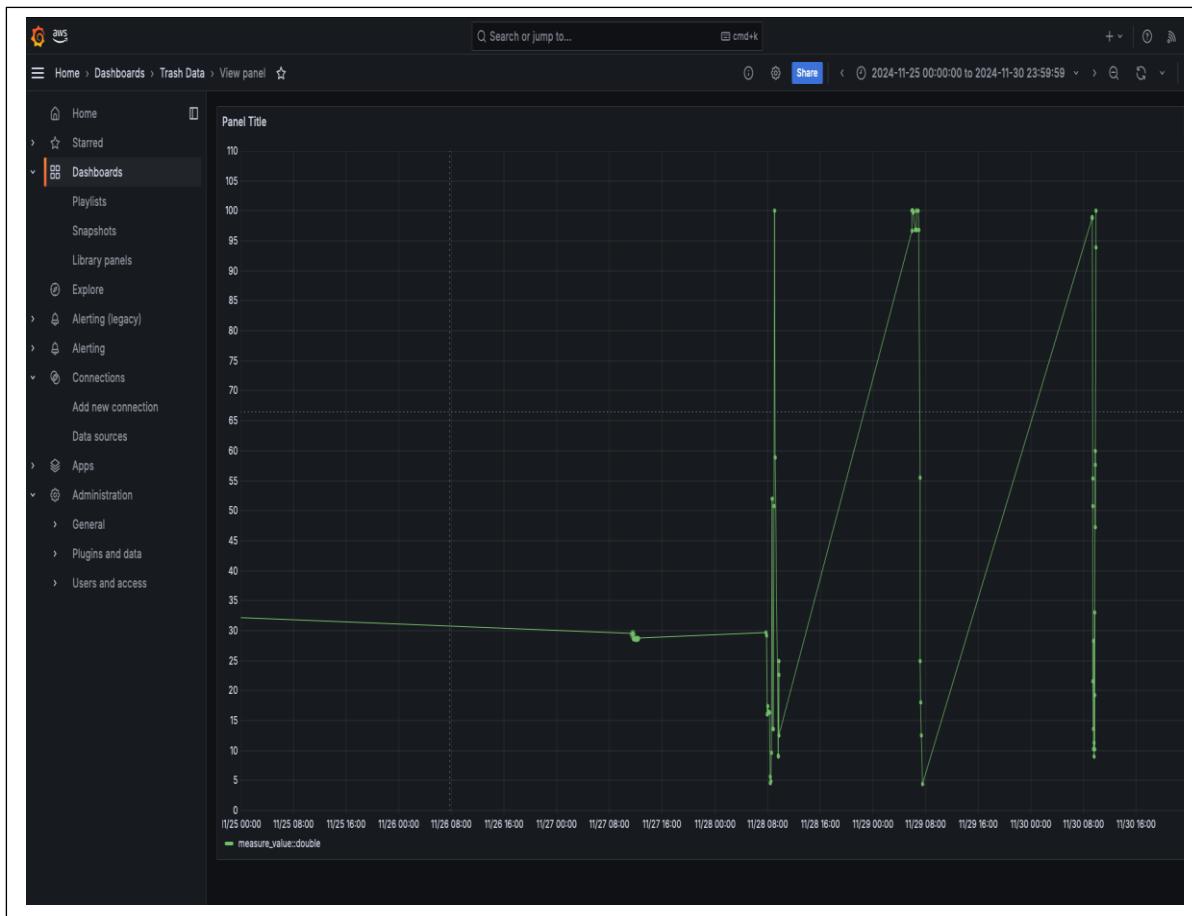
**Bin bin1 - SEV2 - HIGH Capacity Alert**  
trash-data <no-reply@sns.amazonaws.com>  
To: [REDACTED]  
Yesterday at 5:44 PM  
Bin Capacity Alert 🔞  
Bin ID: bin1  
Current Distance: 5.61 cm  
Remaining Capacity: 19.34%  
Filled Capacity: 80.66%  
Severity: SEV2 - HIGH

trash-data
Bin bin1 - SEV2 - HIGH Capacit... Yesterday
☒ Bin Capacity Alert ☒ Bin ID: bin1 Curre...
trash-data
Bin bin1 - SEV2 - HIGH Capacit... Yesterday
☒ Bin Capacity Alert ☒ Bin ID: bin1 Curre...
trash-data
Bin bin1 - SEV1 - CRITICAL Ca... Yesterday
☒ Bin Capacity Alert ☒ Bin ID: bin1 Curre...
trash-data
Bin bin1 - SEV2 - HIGH Capacit... Yesterday
☒ Bin Capacity Alert ☒ Bin ID: bin1 Curre...
trash-data
Bin bin1 - SEV3 - MEDIUM Cap... Yesterday
☒ Bin Capacity Alert ☒ Bin ID: bin1 Curre...
trash-data
Bin bin1 - SEV2 - HIGH Capacit... Yesterday
☒ Bin Capacity Alert ☒ Bin ID: bin1 Curre...
trash-data
Bin bin1 - SEV3 - MEDIUM Cap... Yesterday
☒ Bin Capacity Alert ☒ Bin ID: bin1 Curre...

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

- 1. 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.
- 2.** 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.
- 3. 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 on working for Back-End:**

I spent nearly ~120 hours on this in the past 1 month.