UNIVERSITI TUNKU ABDUL RAHMAN

ACADEMIC YEAR 2022/2023

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY (PERAK)

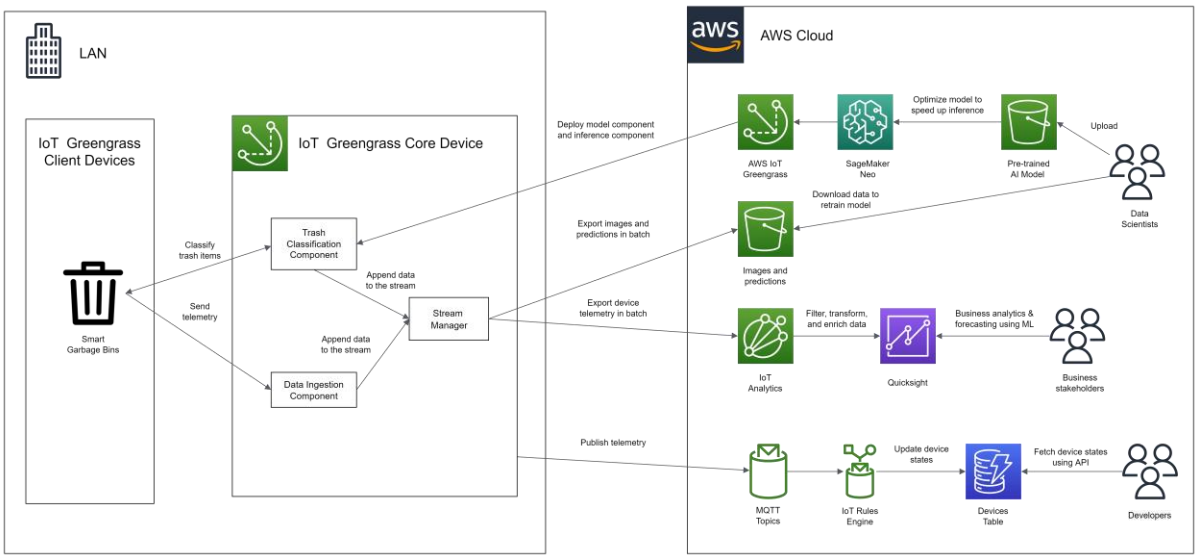
UCCD3113 DISTRIBUTED COMPUTER SYSTEMS

TOPIC: SMART GARBAGE COLLECTION SYSTEM

Name	Student ID	Course
Yap Jheng Khin	1800224	CS
Gan Win Sian	1702594	IA
Cheow Yue Chen	1902365	IA
Kesavan a/l Muniandy	1700867	IA

Abstract

The manual explains the step-by-step approach to implement the architecture shown below. The source code can be found at the *artifacts* directory.



Given the following hypothetical scenarios, International IT Solution Kampar Sdn. Bhd. is tasked to implement a smart garbage collection system for Danish House Student Hostel in Kampar. Below are the system requirements:

1. A smart garbage bin is installed in each floor in every three-floor high rental building.
2. Each rental building has a unique house number (e.g., 1062).
3. Each rental building belongs to one of the house areas such as Beijing, Oxford, Manchester and so on.
4. Every smart garbage bin/client device can internally classify and sort the trash item by requesting the trash classification inference from the core device.
5. A cloud storage solution is set up to store the device's latest fill level. Then, the smart garbage bins send the fill level telemetry on an hourly basis to update the records in the cloud.
6. A core device/edge gateway is set up to process inference request and classify trash.
7. The core device aggregates fill level telemetry from the client device before sending to the cloud for further analysis.
8. The core device collects received trash images and model outputs to the cloud for model retraining.

If the user only concerns about deploying the solution as it is, kindly follow the instructions from Section 1 to Section 6 in sequence. The Section 7 is meant for developers who want to implement a customized solution, or simply want to understand in depth on how the communication between the smart garbage bin, Greengrass components in the core device, and AWS IoT Core works.

It is assumed that the readers have read the core concepts and go through some quick-start tutorials for every AWS cloud services that are used in the solution. These AWS cloud services include:

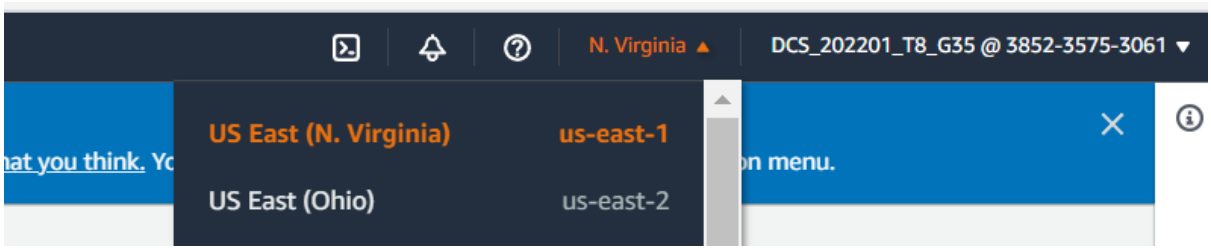
a) Amazon DynamoDB
b) Amazon QuickSight
c) Amazon S3
d) Amazon SageMaker Neo
e) AWS Identity and Access Management (IAM)
f) AWS IoT Analytics
g) AWS IoT Greengrass
h) AWS IoT Rules Engine

Table of Contents

Section 1: General Setup

1.1 AWS Console

1) In the AWS Management Console, ensure that the selected region is US East (N. Virginia). The nearest region is not chosen since the Asia Pacific regions do not fully support all IoT services like IoT Analytics.



1.2 Setup AWS CLI

1) AWS CLI is required to be installed in the core device's virtual machine. The installation instruction can be found here. For windows, run the command below to check the installation of AWS CLI. Add the AWS CLI to Windows path if the installation is not detected.

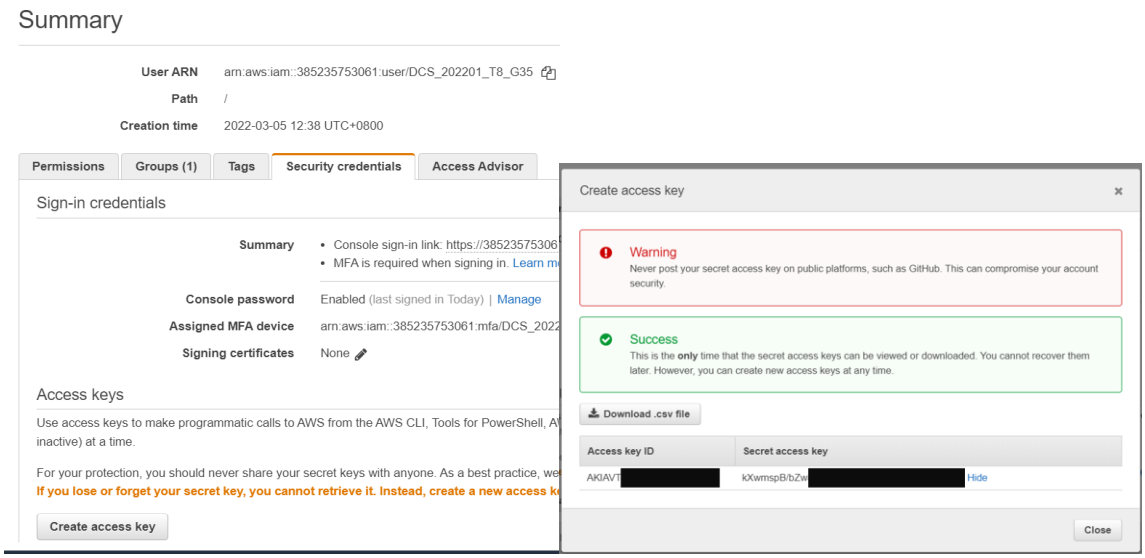
```
1.  aws --version
```

```
C:\Users\User>aws --version
aws-cli/2.4.23 Python/3.8.8 Windows/10 exe/AMD64 prompt/off

C:\Users\User>
```

2) Configure the AWS CLI to run AWS commands on behalf of your IAM account. In the **IAM console** navigation menu, choose **Users**. Choose the name of the user that you want to create the access key from. The username chosen in this manual is DCS_202201_T8_G35. For security reasons, try not to create access keys from root account but from IAM user account.

3) In the user summary page, choose **Security credentials** tab. Choose **create access key** under Access keys section. Save the access key ID and secret access key to a secure place before closing the window.



4) For Windows, run the command below to configure the AWS CLI. Enter the new access key and secret access key into the console. Follow the inputs shown in the image below.

```
1.  aws configure
```

1.3 Setup Virtual Machines

1) There is no hardware to implement the system.

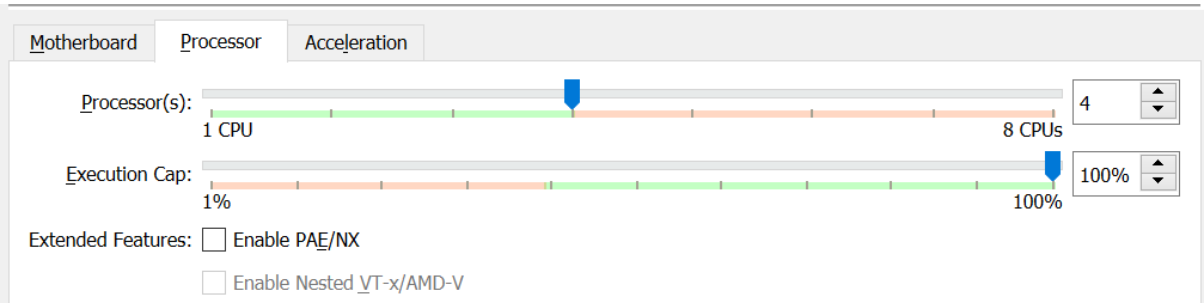
2) The virtualization software used is Oracle VM VirtualBox Version 6.1.32. The virtual machines for core device and client device use 64-bit Ubuntu 18.04.6. The ISO image can be downloaded from https://releases.ubuntu.com/18.04. Feel free to install the latest 64-bit Ubuntu and adjust the bash script when necessary.

3) Throughout the manual, to avoid confusion on which device does a screenshot belongs to, look at the <user>@<host-name> at the start of each line in the terminal. **danish-garbage-bin-1@danish-garbage-bin-1** is referring to the client device running on Ubuntu. **jhengkhinyap@danish-gcs-debian-core** and **dcs_t8_g35@dcs-202201-t8-g35** is referring to the core device running on Debian and Ubuntu, respectively. There are some screenshots from Debian's terminal since the author originally tested software installation on Debian but switch to Ubuntu 18.04.6 as demanded by some Greengrass public components like this one.

Ubuntu Requirement for Core Device and Client Device

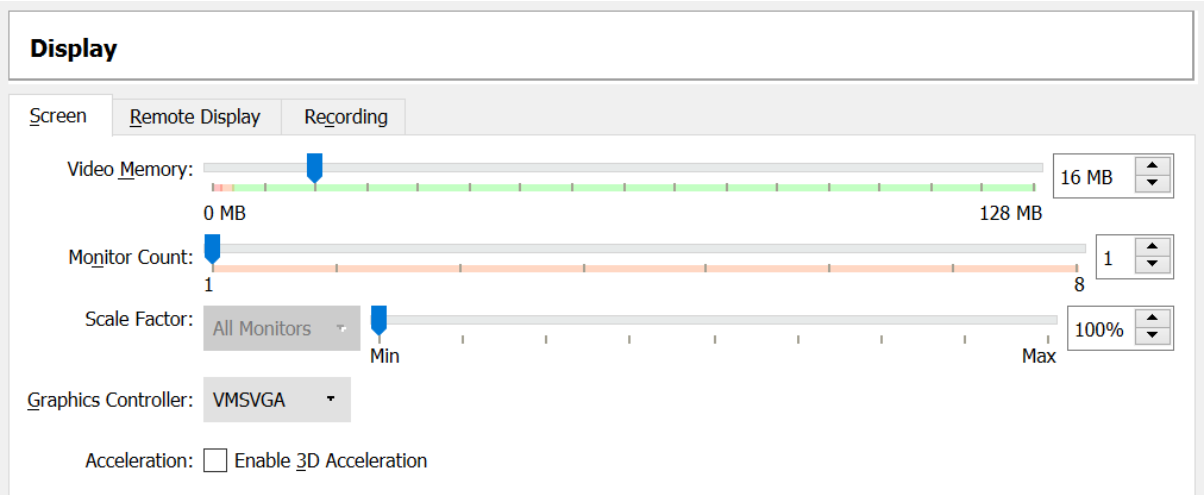
1) The optimal resource requirements of the Ubuntu 64-bit virtual machine for core device and client device are shown in the image form. Adjust the requirement accordingly based on your host OS and hardware specifications.

2) The minimum required processor is one.

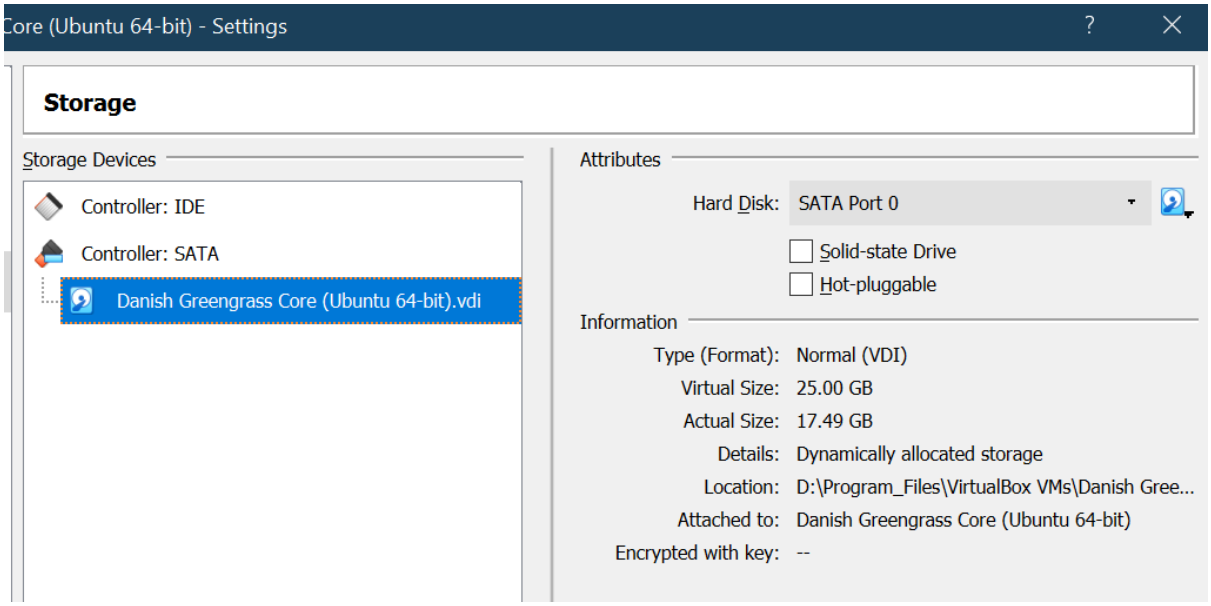


3) The minimum required RAM is 2048MiB.

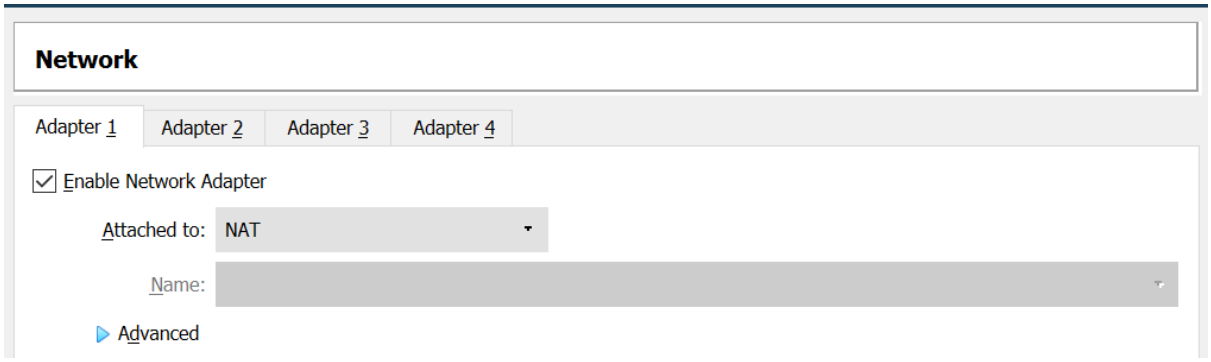
4) Then minimum required video memory is 16MiB.

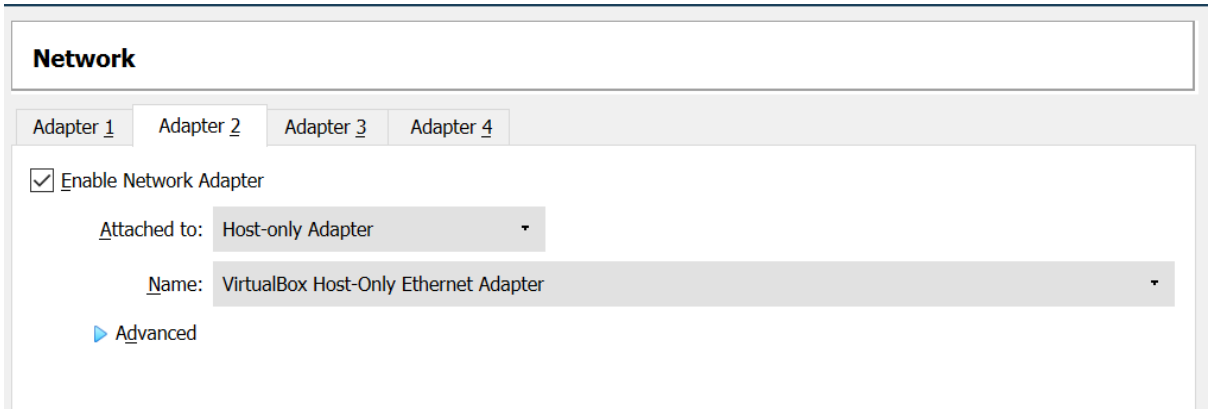


5) The virtual disk image format is VDI while its type is dynamically allocated. Allocate a minimum of 12GiB for the virtual disk image. It is assumed that the minimal installation option is selected while installing the OS, which means only essential Ubuntu packages and web browser are pre-installed, thus saving some storage space.



6) Below is the network configuration for the core device. More details are discussed in the section below.

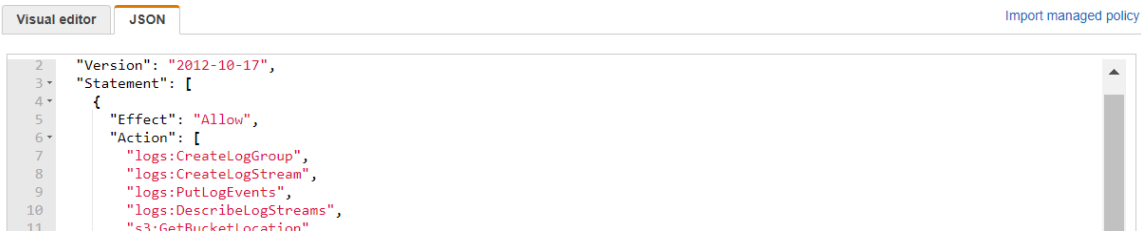




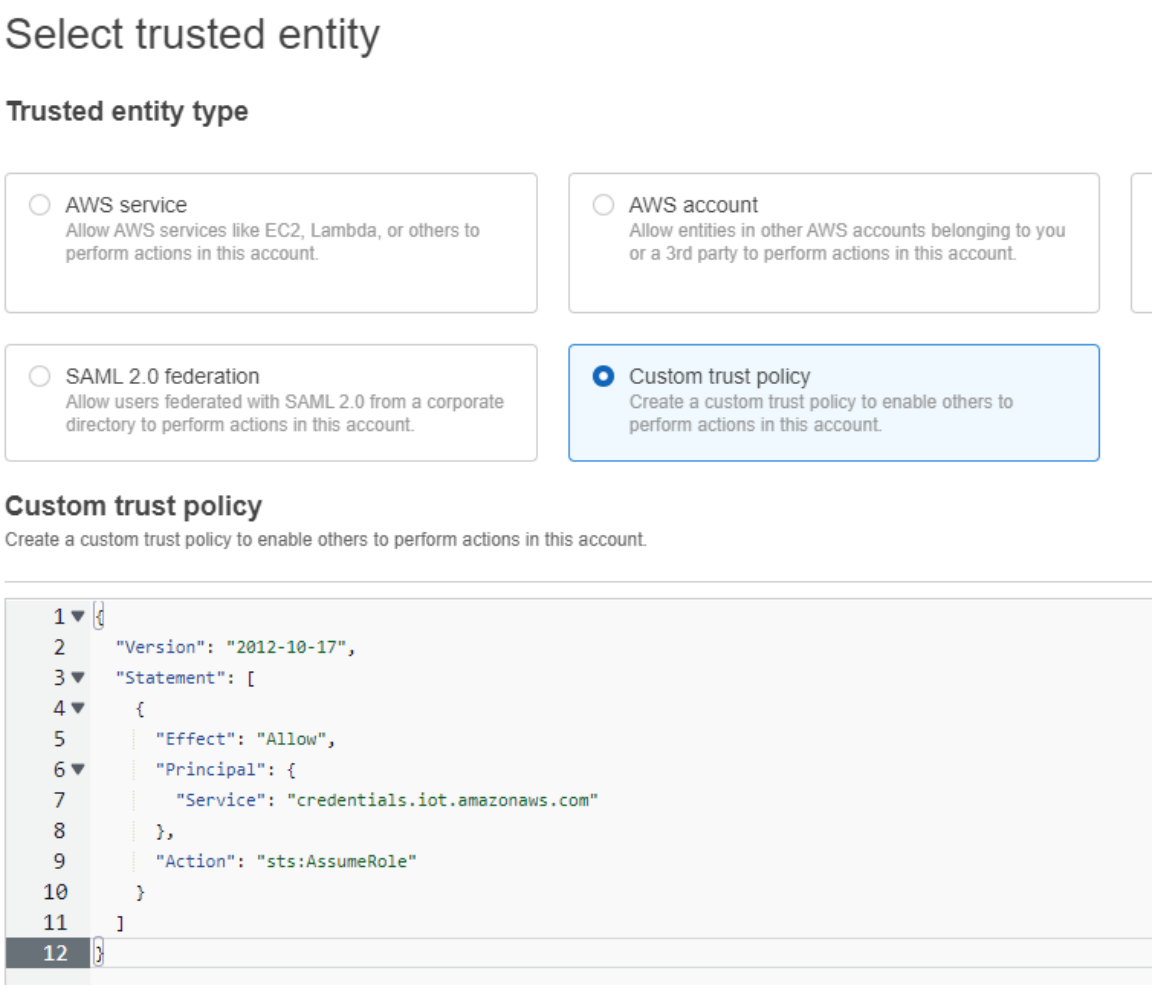
Section 2: Provisioning AWS Cloud Resource

2.1 IAM Roles and Policies

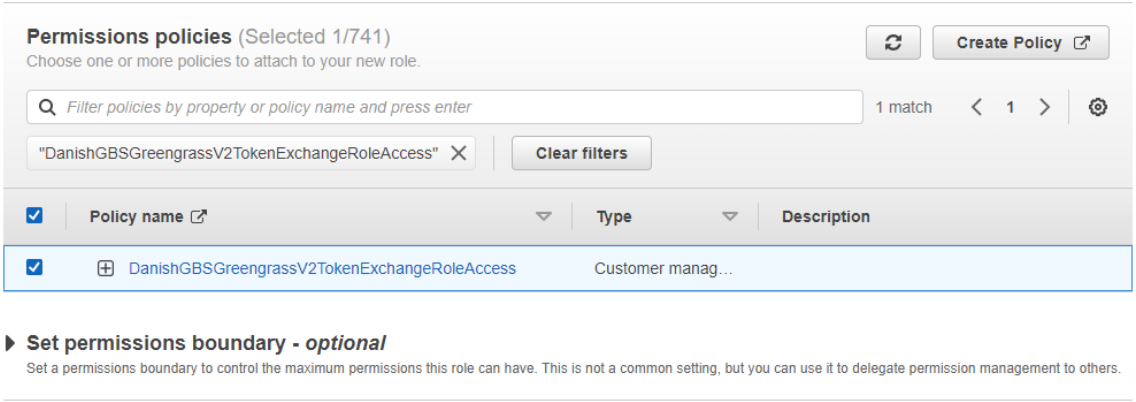
1) In the **IAM Console** navigation menu, select **Policies** > **Create policy**. Copy the permission policy's content from this file to the editor in the **JSON** tab. Select **Next: Tags** > **Next: Review**. Name the policy accordingly before selecting **Create policy**.



2) In the IAM Console navigation menu, select **Roles** > **Create role**. Select **Custom trust policy**. Copy the custom trust policy's content from this file to the editor as shown below. Select **Next**.



3) Attach both the customer managed and AWS managed permission policies to the new role as instructed in the following section. Click **Next**. Name the role accordingly before selecting **Create role**.



2) Repeat the steps for remaining service roles below.

2.1.1 Service Role #1: DanishGBSGreengrassV2TokenExchangeRole

Description: A token exchange role that authorizes core devices to download component artifacts from S3 bucket, export images and model outputs to the S3 bucket, and finally export fill level telemetry to the IoT Analytics channel.

- Trusted entity type: Custom trust policy
- Custom trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- Permission policy #1 name: **DanishGBSGreengrassV2TokenExchangeRoleAccess**
- Permission policy #1:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
```

```
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": ["iotanalytics:BatchPutMessage"],
      "Resource": [
        "arn:aws:iotanalytics:us-east-
1:385235753061:channel/danish_gcs_bin_telemetry_iot_analytics_channel"
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["s3:PutObject"],
      "Resource": [
        "arn:aws:s3:::danish-gcs-image-and-model-output-bucket/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": [
        "arn:aws:s3:::danish-gcs-model-artifacts-
bucket/v*/variant.DLR.artifacts/*",
        "arn:aws:s3:::danish-gcs-model-artifacts-bucket/v*/neo-compiled/*",
        "arn:aws:s3:::danish-gcs-model-artifacts-
bucket/v*/aws.greengrass.DLRImageClassification.artifacts/*",
        "arn:aws:s3:::danish-gcs-model-artifacts-
bucket/v*/danish.GCS.TelemetryIngestion/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "arn:aws:s3:::danish-gcs-image-and-model-output-bucket/*"
      ]
    }
  ]
}
```

2.1.2 Service Role #2: DanishGCSSageMakerIAMRole

Description: Authorize Amazon SageMaker to access related AWS services like S3 bucket, and to download and upload model artifacts to and from the S3 bucket.

- Trusted entity type: Custom trust policy
- Custom trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- Permission policy #1 name: DanishGCSSageMakerAccessS3Policy
- Permission policy #1:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::danish-gcs-model-artifacts-bucket/*"
      ]
    },
    {
      "Action": ["s3:GetObject"],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::danish-gcs-model-artifacts-bucket/v*/pytorch/*"
      ]
    },
    {
      "Action": ["s3:PutObject"],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::danish-gcs-model-artifacts-bucket/v*/neo-compiled/*"
      ]
    }
  ]
}
```

- Permission policy #2 name: AmazonSageMakerFullAccess
- Permission policy #2: AWS-Managed

2.1.3 Service Role #3: DanishGCSBulkRegistrationAccessS3Role

Description: Provide S3 bucket access to retrieve parameter values to be used in the registration template during bulk registration.

- Trusted entity type: Custom trust policy
- Custom trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- Permission policy #1 name: **DanishGCSBulkRegistrationAccessS3Policy**
- Permission policy #1:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": [
        "arn:aws:s3:::danish-gcs-bulk-reg-params-bucket/*"
      ]
    }
  ]
}
```

- Permission policy #2 name: **AWSIoTThingsRegistration**
- Permission policy #2: AWS-Managed

2.1.4 Service Role #4: DanishGCSDescribeThingRole

Description: Allow AWS IoT Analytics access to retrieve device metadata for enriching the telemetry received from the core device.

- Trusted entity type: Custom trust policy
- Custom trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotanalytics.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- Permission policy name: **DanishGCSDescribeThingPolicy**
- Permission policy:

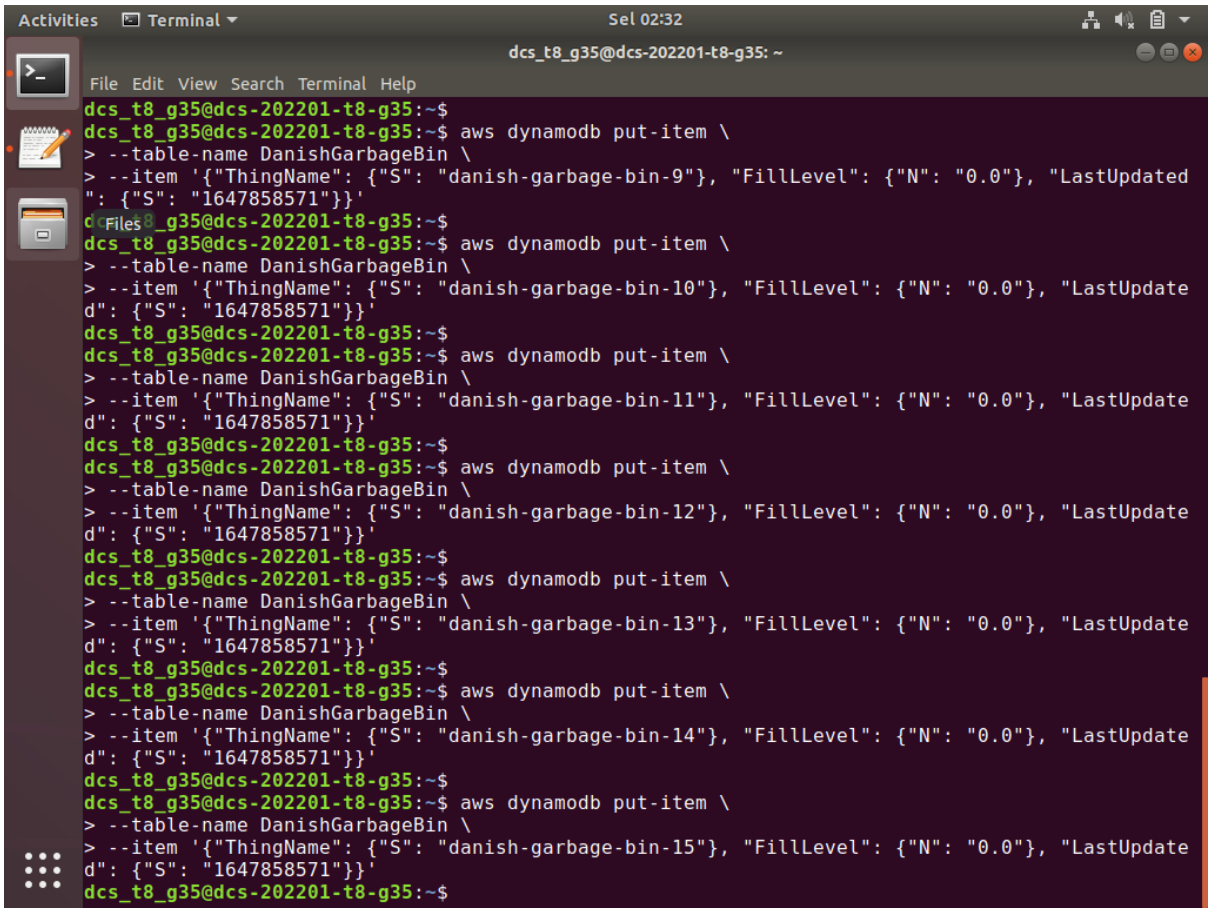
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["iot:DescribeThing"],
      "Resource": [
        "arn:aws:iot:us-east-1:385235753061:thing/danish-garbage-bin-*"
      ]
    }
  ]
}
```

2.2 Amazon DynamoDB

Note: The commands can be found in *aws_cli/dynamodb_commands.txt*.

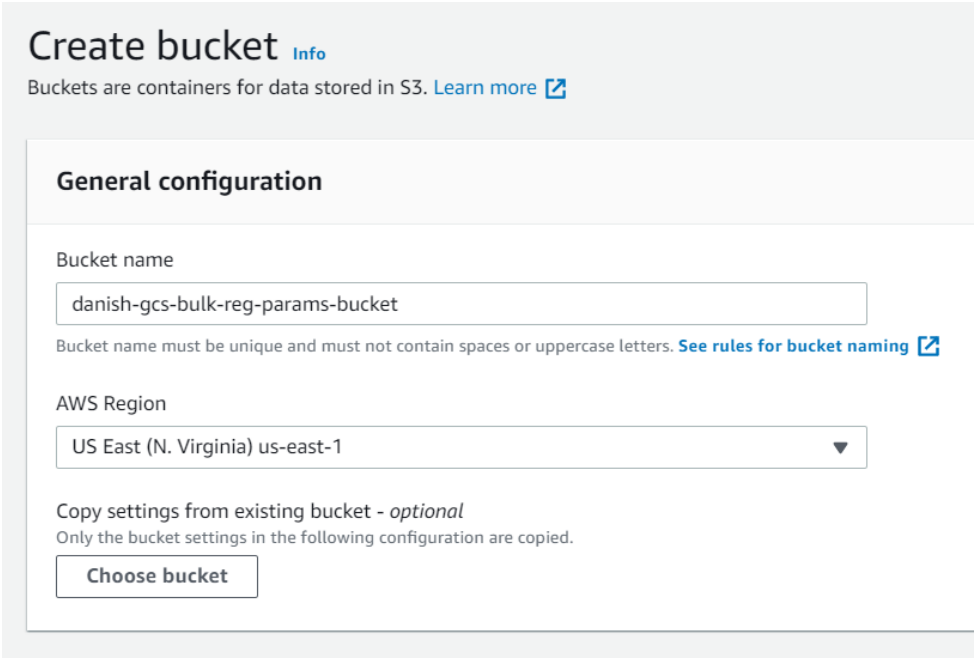
1) Create DynamoDB table named **DanishGarbageBin**.

2) Insert device records into **DanishGarbageBin**.



2.3 Amazon S3

1) In the **Amazon S3 Console** navigation menu, select **Buckets** > **Create bucket**. Leave other settings as default and select **Create bucket**.



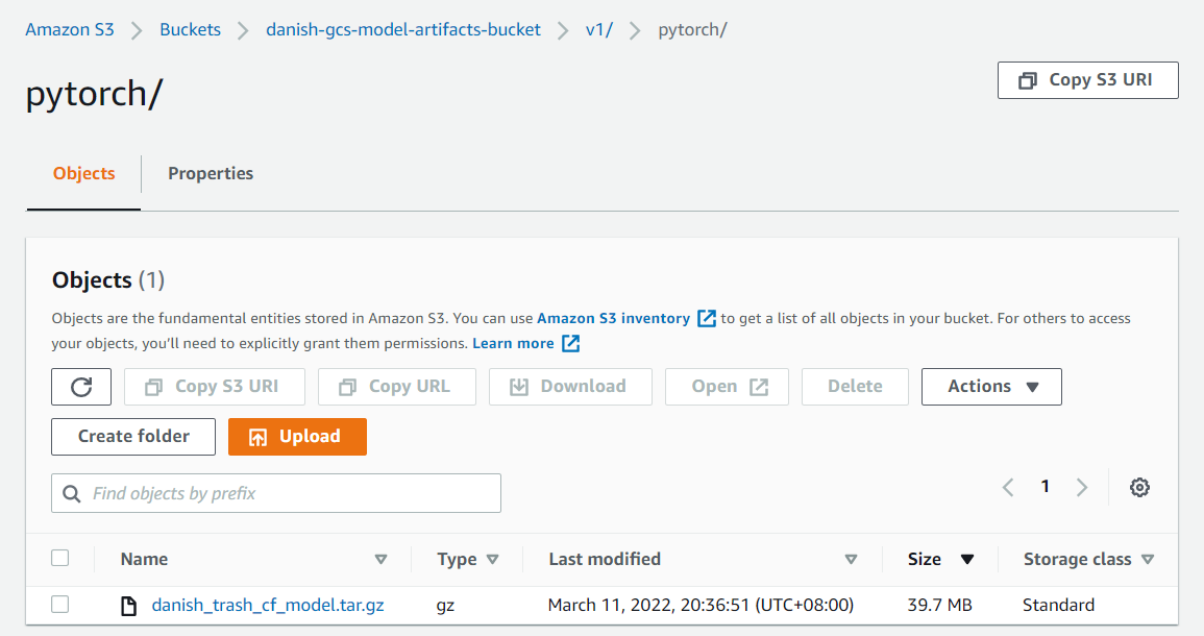
2) Repeat the steps above for creating the S3 buckets listed below.

Bucket Name	Description
danish-gcs-bulk-reg-params-bucket	Store the parameter values of the provisioning template that is used in bulk registration.
danish-gcs-image-and-model-output-bucket	Store the AI model inputs and outputs. The AI model inputs include device-captured images; the AI model outputs include the predicted trash types and the prediction confidence scores.
danish-gcs-model-artifacts-bucket	Store the Greengrass component artifacts that fetched by the core device during installation. Store the PyTorch trash classifier and neo-compiled trash classifiers.

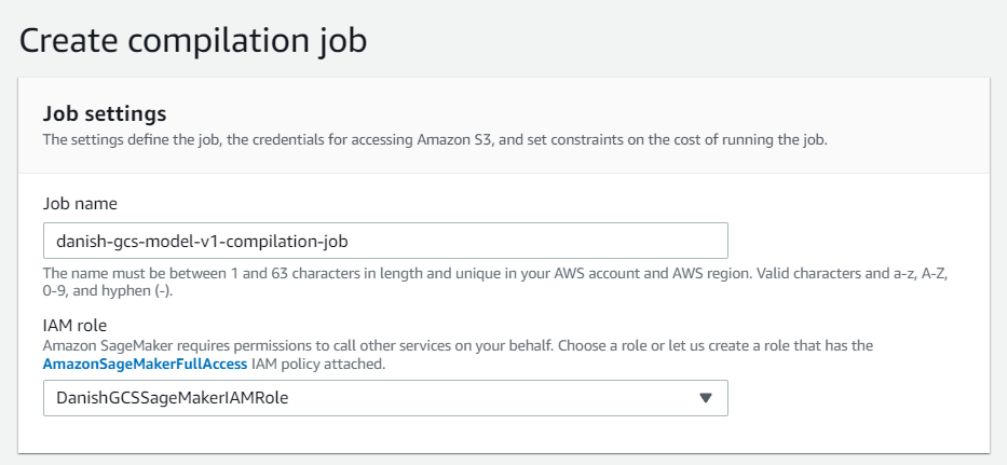
2.4 Amazon SageMaker Neo Compilation

1) Amazon SageMaker Neo model compilation can ensure any model to optimally perform inference across multiple cloud instances and edge devices regardless of different hardware and software specification (Amazon Web Services, Inc., 2022b). In this case, SageMaker Neo automatically optimizes the PyTorch trash classifier for inference on Linux machines running on Intel processors.

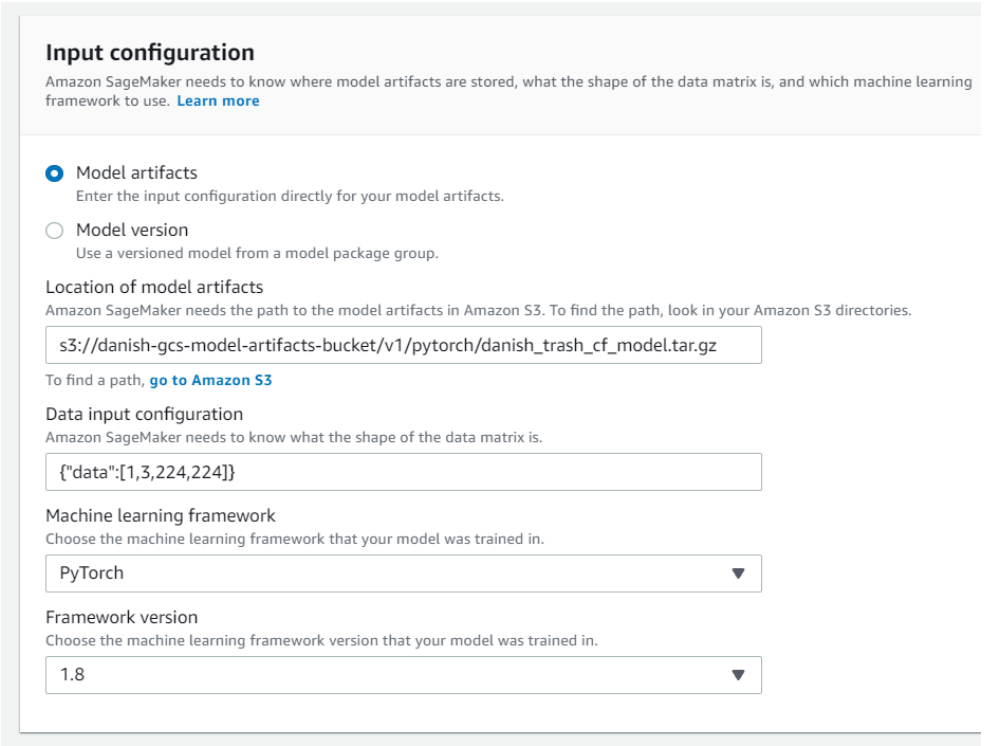
2) Before proceeding, ensure that the (1) PyTorch model artifacts is uploaded to **danish-gcs-model-artifacts-bucket/v1/pytorch** folder and (2) DanishGCSSageMakerIAMRole role is created.



3) In the **Amazon SageMaker** navigation menu, select **Inference** > **Compilation jobs** > **Create compilation job**. Name the compilation job as **danish-gcs-model-v1-compilation-job**. Choose the **DanishGCSSageMakerIAMRole** in the IAM role drop-down menu.

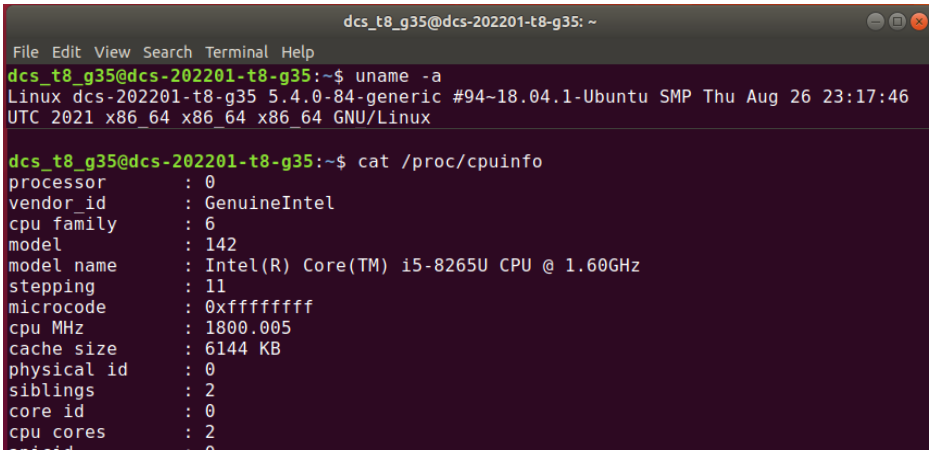


3) Under the **Input configuration**, enter the following information as shown in the image below. The **Data input configuration** inform the compiler that the model expects an RGB image of width of 224 pixels and height of pixels. Refer to the official documentation to configure non-PyTorch models (Amazon Web Services, Inc., 2022f).

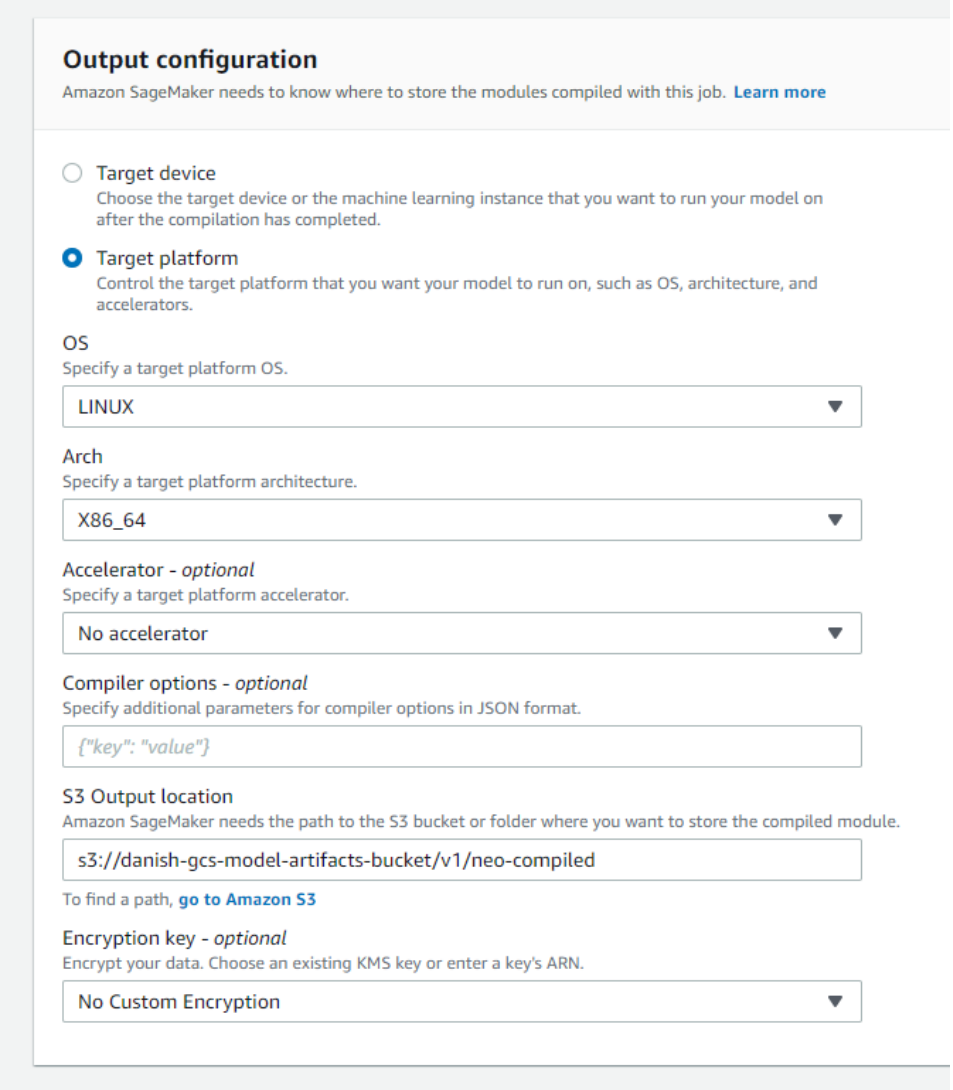


4) Execute the following commands in Linux to check the target platform OS and the target architecture. The information is needed in the **Output configuration** section.

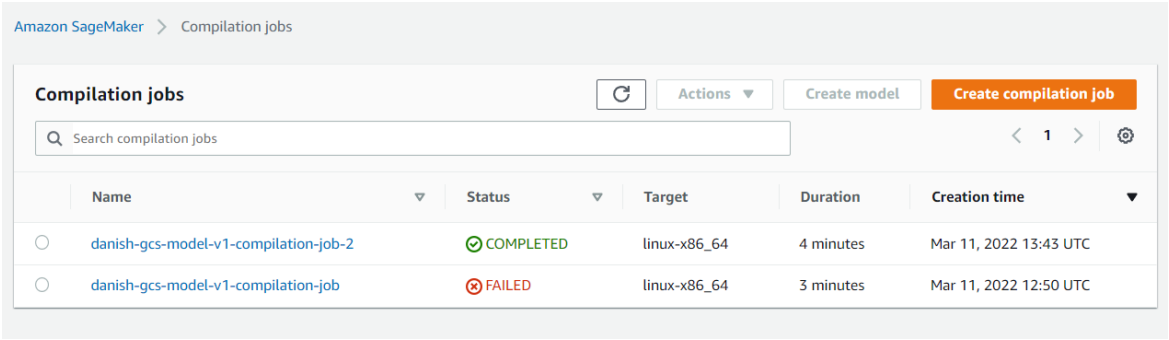
```
uname -a
cat /proc/cpuinfo
```



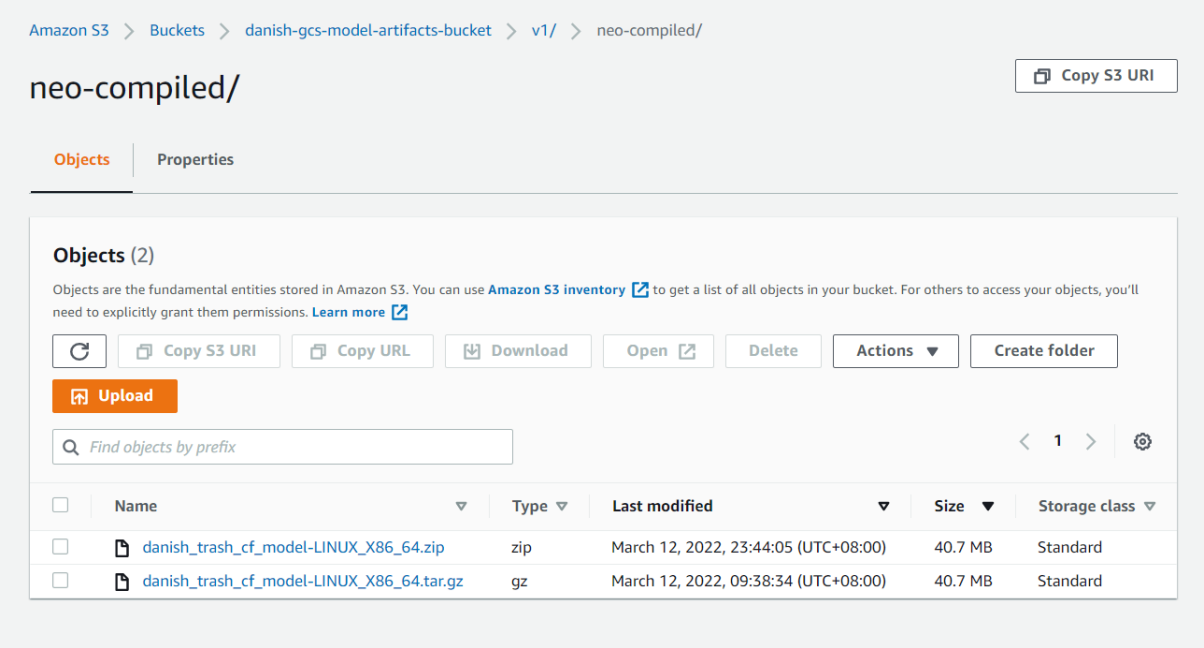
5) Under the **Output configuration**, select **Target platform** and enter the information as shown in the image below. Ensure that the **neo-compiled** folder is created in the S3 bucket.



6) Select **Submit**. Wait for the compilation job to be completed.



7) If the compilation job has completed, navigate to the output folder, in this case, **neo-compiled** folder to download the .tar.gz file. Extract the file and re-compress the file in ZIP format and upload back to the same folder. This is because the Greengrass component installer does not support .tar.gz decompression but supports ZIP decompression.



2.5 AWS IoT Core Resources

In the AWS CLI:

1) Execute the commands below to create the thing types that are required for the client device and core device.

```
aws iot create-thing-type --thing-type-name "danish-gcs-core" \
--thing-type-properties "thingTypeDescription=Core device(s) for garbage
collection system in Danish house"
```

```
aws iot create-thing-type --thing-type-name "danish-garbage-bin" \
--thing-type-properties "thingTypeDescription=Client device(s) for garbage
collection system in Danish house"
```

2) Execute the commands below to create thing policy for the core device. The policy can be found in *aws_cli/DanishGCSCoreThingPolicy.json*.

```
aws iot create-policy --policy-name DanishGCSCoreThingPolicy \
--policy-document file://DanishGCSCoreThingPolicy.json \
--tags Key=System,Value="Garbage collection system"
Key=Customer,Value="Danish house" \
Key=Country,Value=Malaysia
```

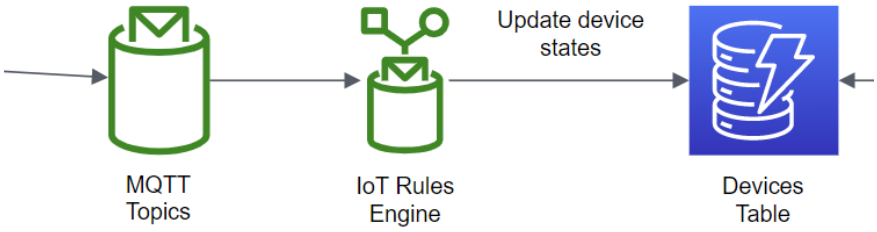
3) Execute the commands below to create thing policy for the client device. The policy can be found in *aws_cli/DanishGCSBinThingPolicy.json.*

```
aws iot create-policy --policy-name DanishGCSBinThingPolicy \
--policy-document file://DanishGCSBinThingPolicy.json \
--tags Key=System,Value="Garbage collection system"
Key=Customer,Value="Danish house" \
Key=Country,Value=Malaysia
```

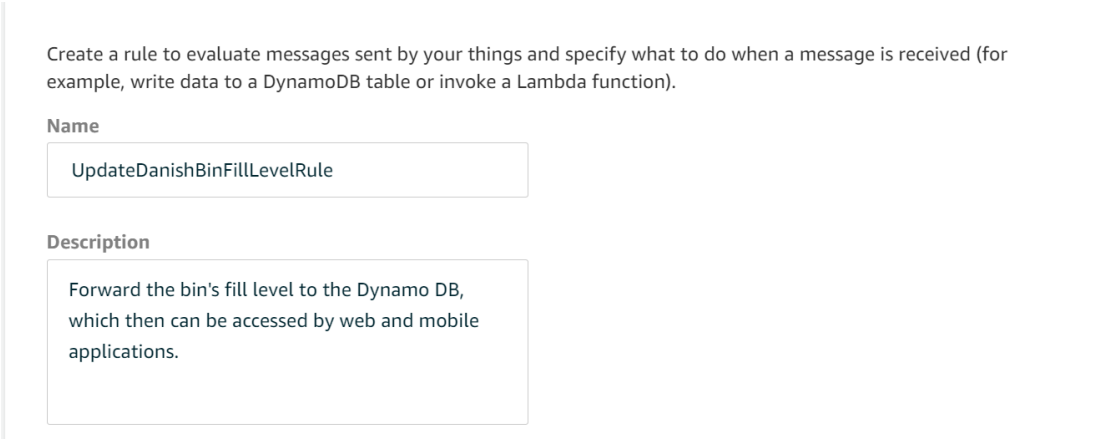
4) Create role alias to point to the token exchange role's ARN. This ensures that only role alias is updated instead of core devices when the role's ARN changes (Amazon Web Services, Inc., 2022d). Ensure that the **DanishGBSGreengrassV2TokenExchangeRole** has created before proceed.

```
aws iot create-role-alias --role-alias
DanishGBSGreengrassCoreTokenExchangeRole \
--role-arn
arn:aws:iam::385235753061:role/DanishGBSGreengrassV2TokenExchangeRole
```

2.6 AWS IoT Rules Engine

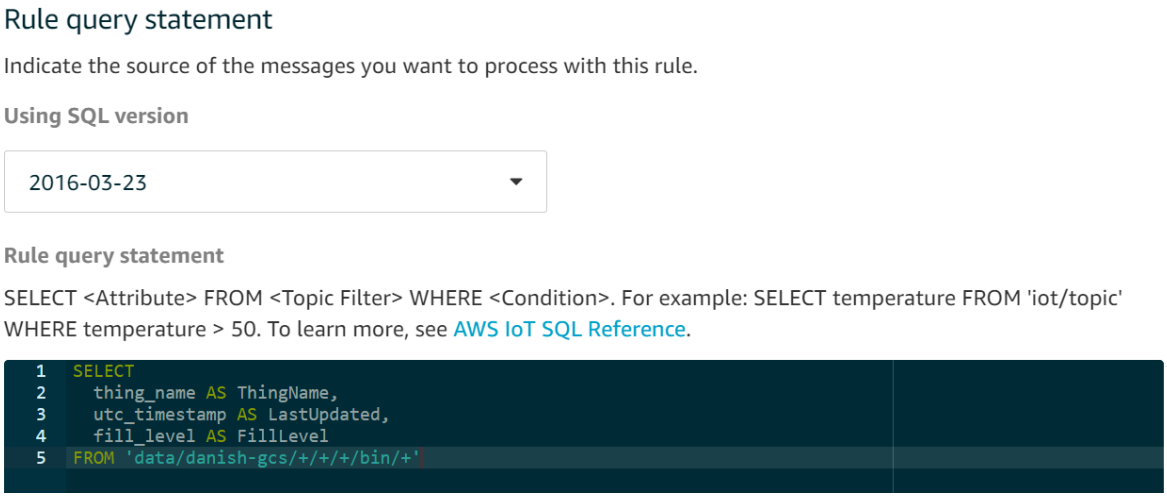


1) In the **AWS IoT Console** navigation menu, select **Act** > **Rules** > **Create**. Name the new rule as **UpdateDanishBinFillLevelRule**.

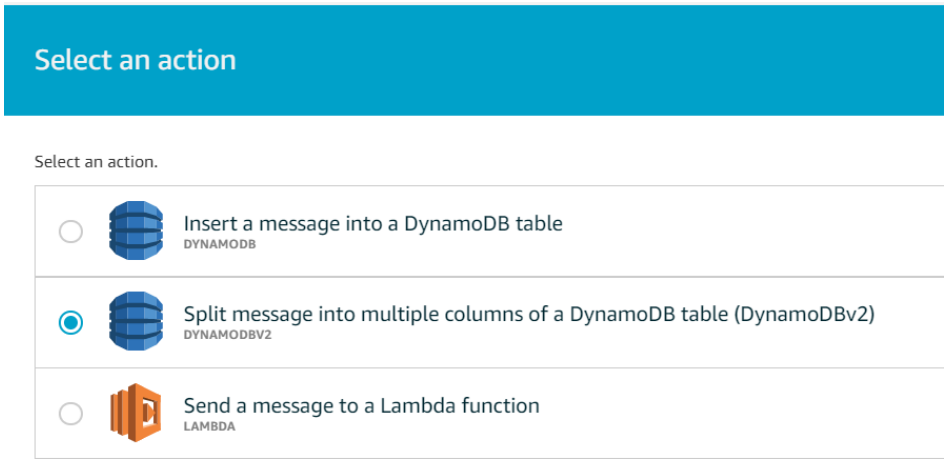


2) Under the **Rule query statement**, set the rule query statement as:

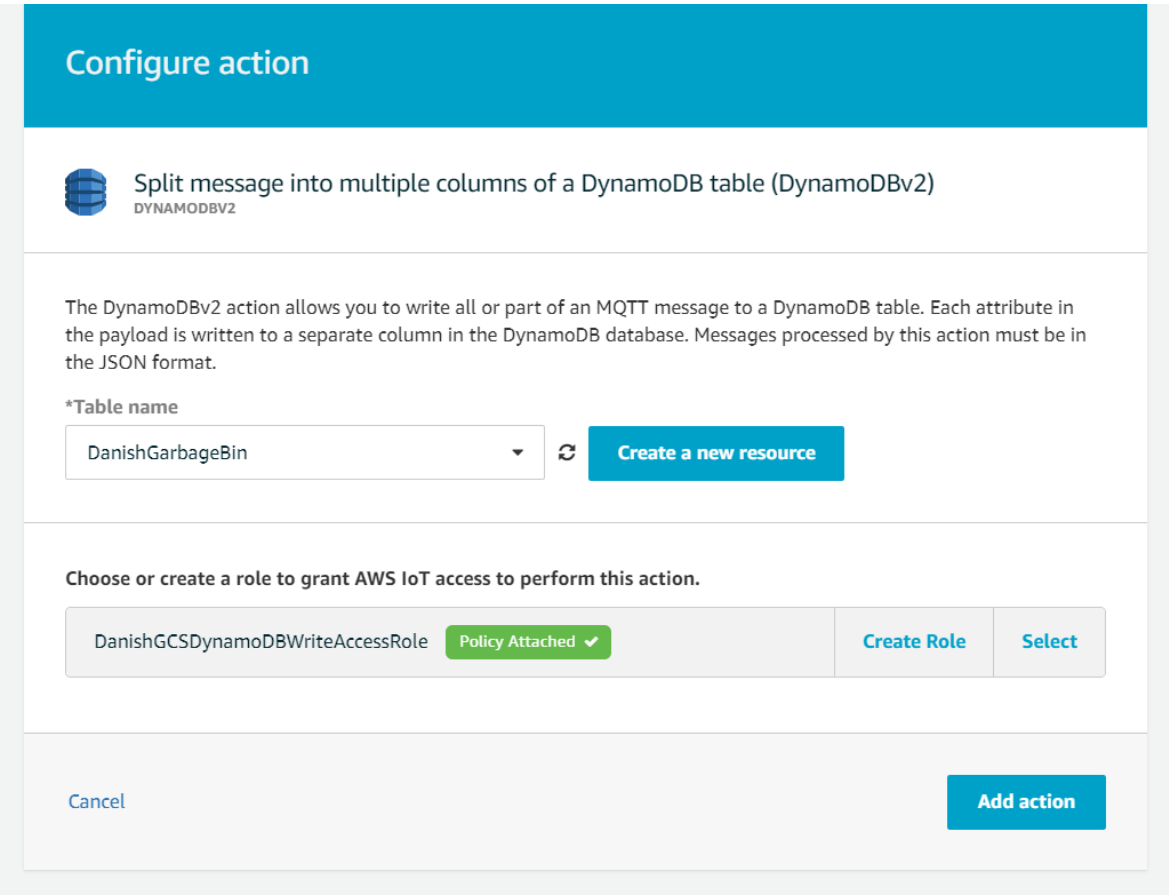
SELECT thing_name **AS** ThingName, utc_timestamp **AS** LastUpdated, fill_level **AS** FillLevel **FROM** 'data/danish-gcs/+/+/+/bin/+'



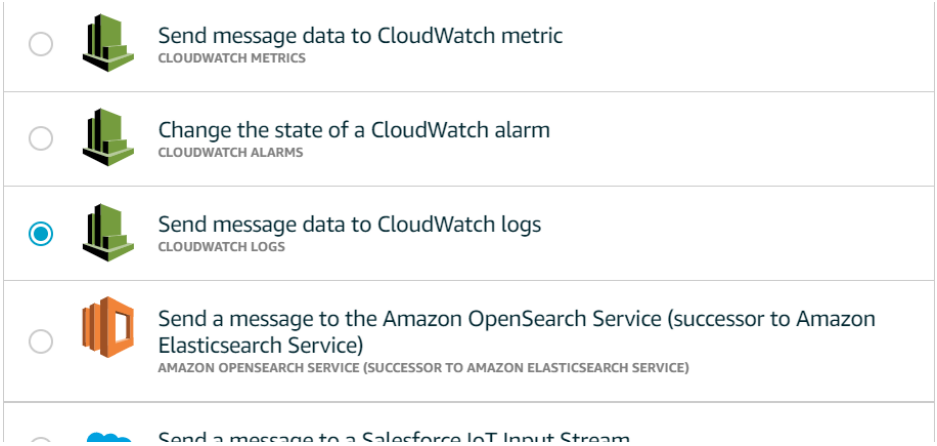
3) Select **Add action** under **Set one or more actions**. Select **DYNAMODBV2** > **Configure action**.



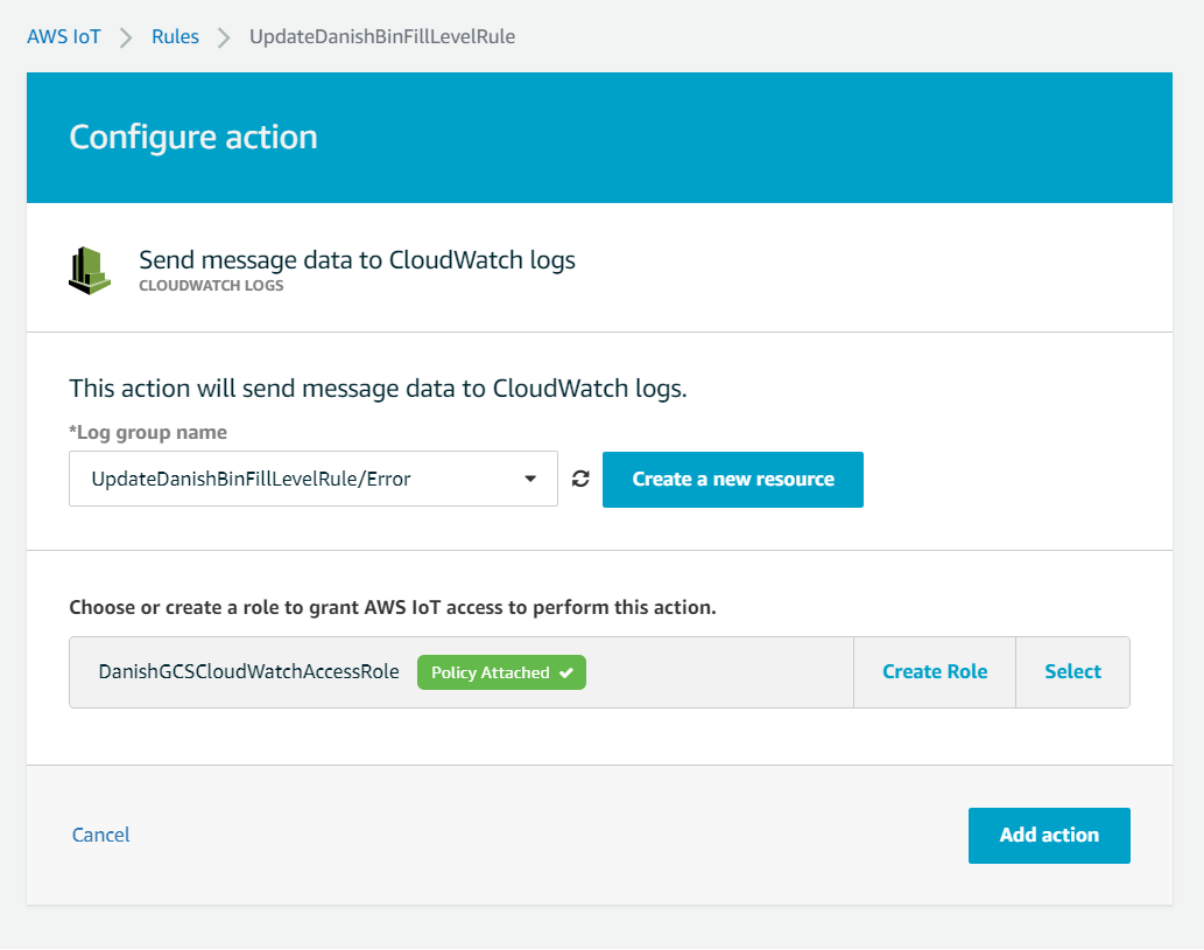
4) As shown in the image below, select the newly create table named **DanishGarbageBin**. Then, select **Create Role**, it will open a pop-up window to prompt the name of the role. Name the role as **DanishGCSDynamoDBWriteAccessRole**. Select **Add action**.



5) Select **Add action** under **Error action**. Select **CLOUDWATCH LOGS** > **Configure action**.



6) As shown in the image below, first, select **Create a new resource**. It will open a page to create the CloudWatch log group. Name the log group as **UpdateDanishBinFillLevelRule/Error** and leave other settings as default. Then, select **Create Role**, it will open a pop-up window to prompt the name of the role. Name the role as **DanishGCSCloudWatchAccessRole**.



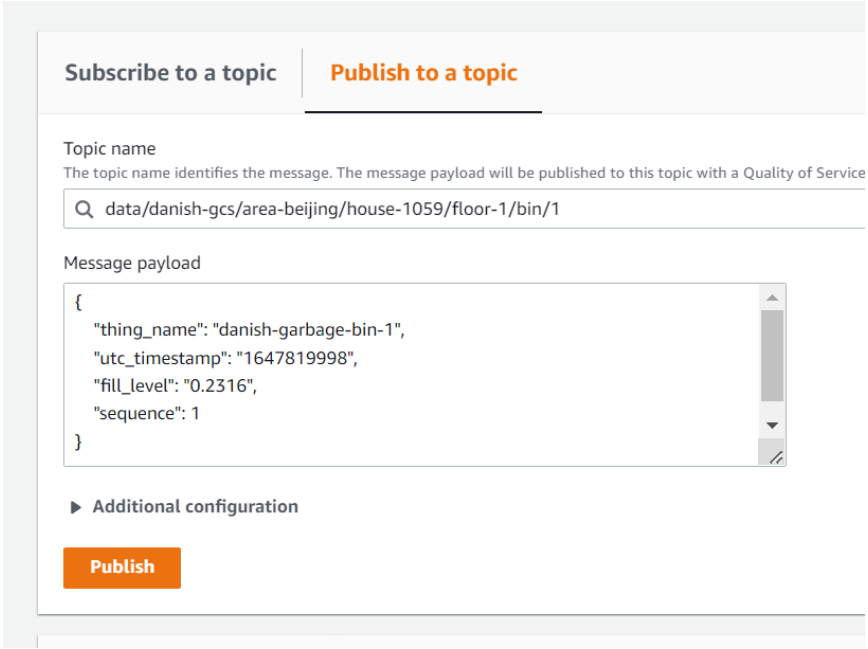
7) Finally, select **Create rule**.

8) To check that the rule is working, in the **AWS IoT Console** navigation menu, select **Test** > **MQTT test client**. Select **Publish to a topic**. Enter the details as shown below before selecting **Publish**.

Topic name: data/danish-gcs/+/+/+/bin/+

Message payload:

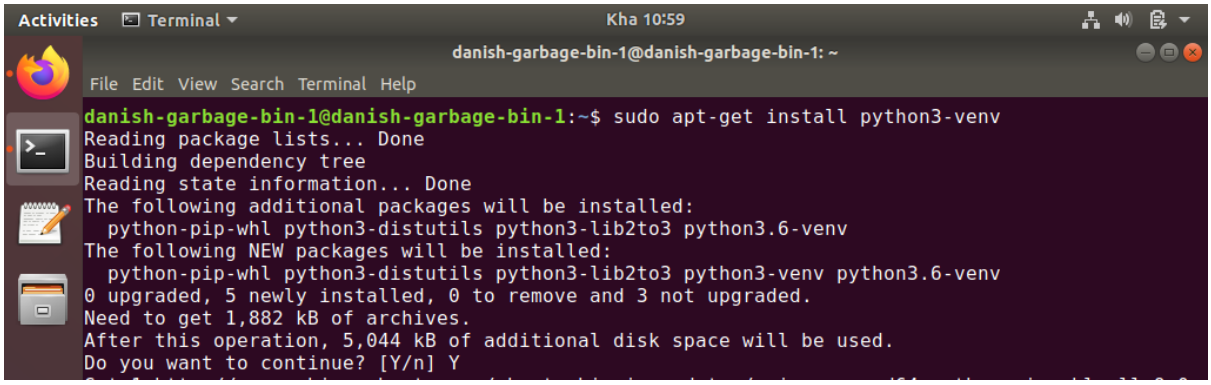
```
{
    "thing_name": "danish-garbage-bin-1",
    "utc_timestamp": "1647819998",
    "fill_level": "0.2316",
    "sequence": 1
}
```



9) In the **DynamoDB Console** navigation menu, select **Tables** > **DanishGarbageBin** > **Explore table items**. Check that the corresponding item has been updated. If not, check the CloudWatch logs for any error.

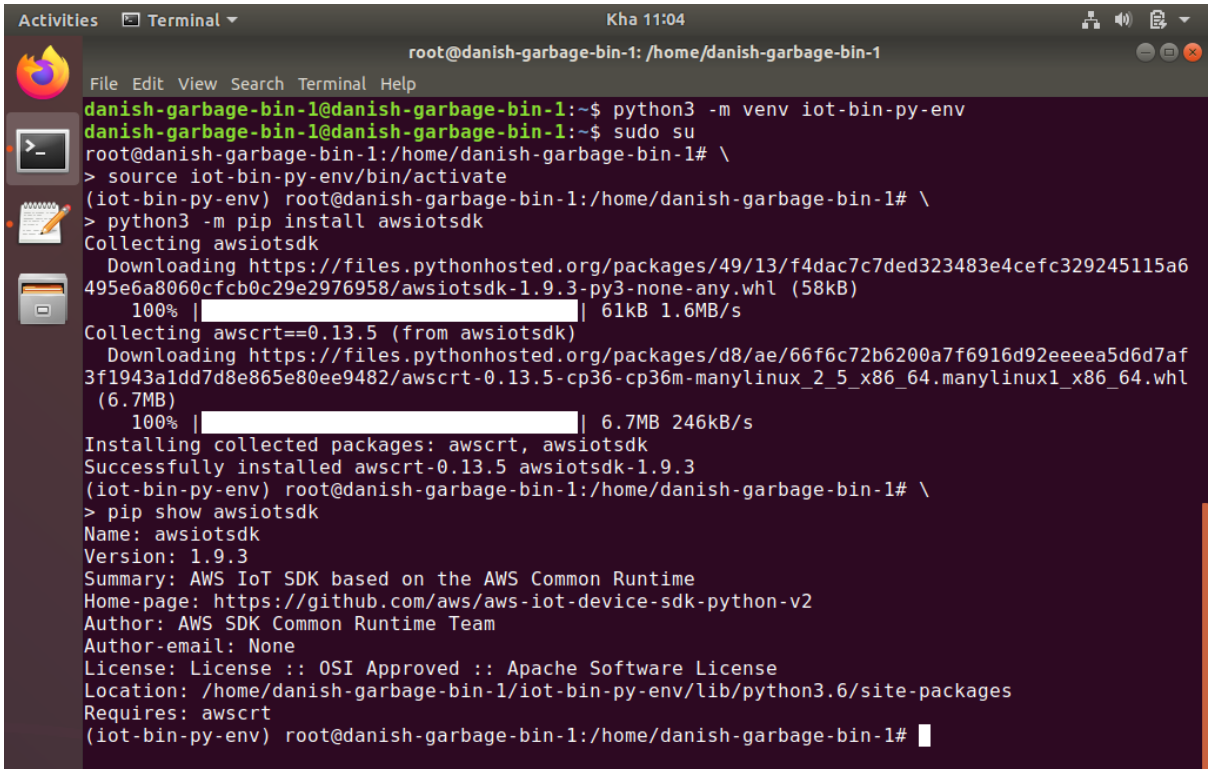
Section 3: Client Device Setup

1) Install **python3-venv** to enable the creation and management of Python virtual environment. Python virtual environments isolate different Python applications into environments, where each environment contains its own Python dependencies isolated from the rest.



2) Create a Python virtual environment named **iot-bin-py-env**. Activate the newly created environment and install a Python package named **awsiotsdk**.

```
python3 -m venv iot-bin-py-env
sudo su
source iot-bin-py-env/bin/activate
python3 -m pip install awsiotsdk
```



3) Run the command below to run the Python files located in *client_device* directory.

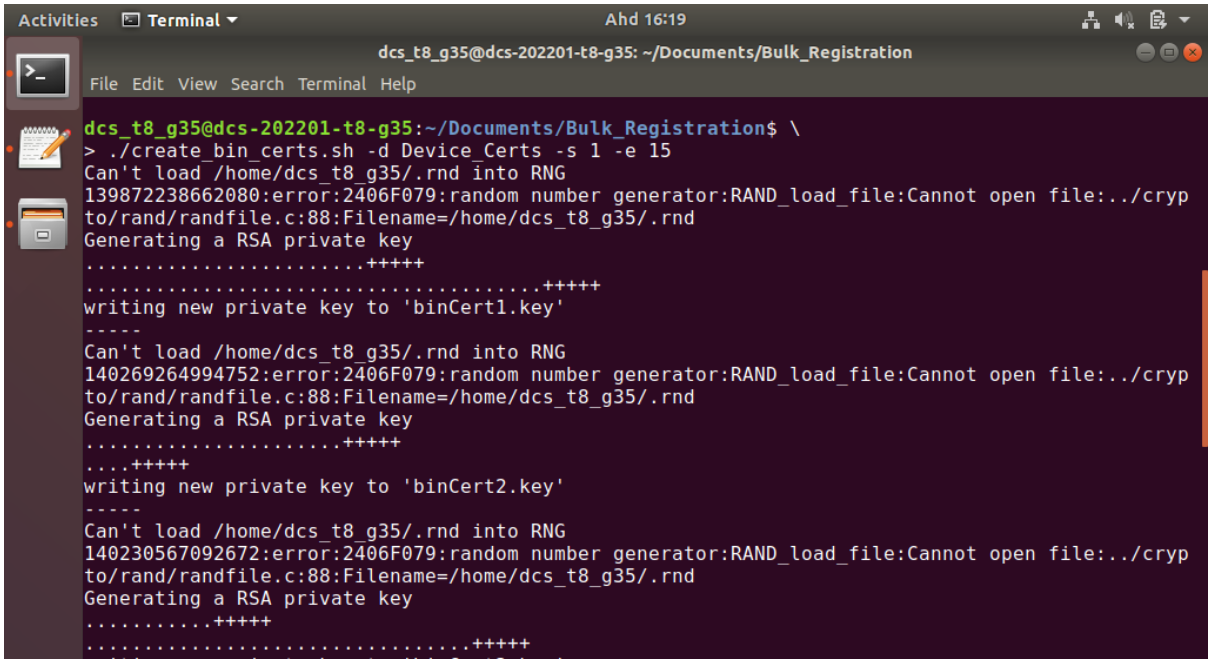
```
sudo su
source iot-bin-py-env/bin/activate
```

3.1 Register Client Devices in Bulk

1) It is cumbersome to manage the creation of AWS IoT thing resource for client devices one by one through the AWS Management Console. Hence, bulk registration is used to provision the cloud resources like thing object, certificate, and policy for the client devices at scale. The source code of the bash scripts can be found at *bulk_registration* folder.

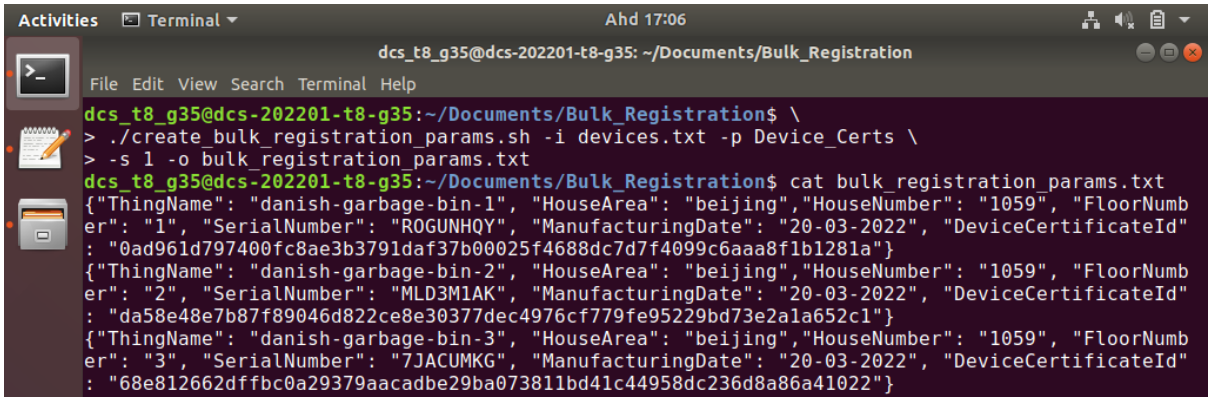
2) Execute the command below to generate new private RSA key for the client devices. The script generates the certificate signing request (CSR) based on the newly generated private RSA key. The **d** argument indicates which directory path to save the key files and CSR files to; the **s** and **e** argument indicate the start ID and the end ID for the device, respectively.

```
./create_bin_certs.sh -d Device_Certs -s 1 -e 15
```

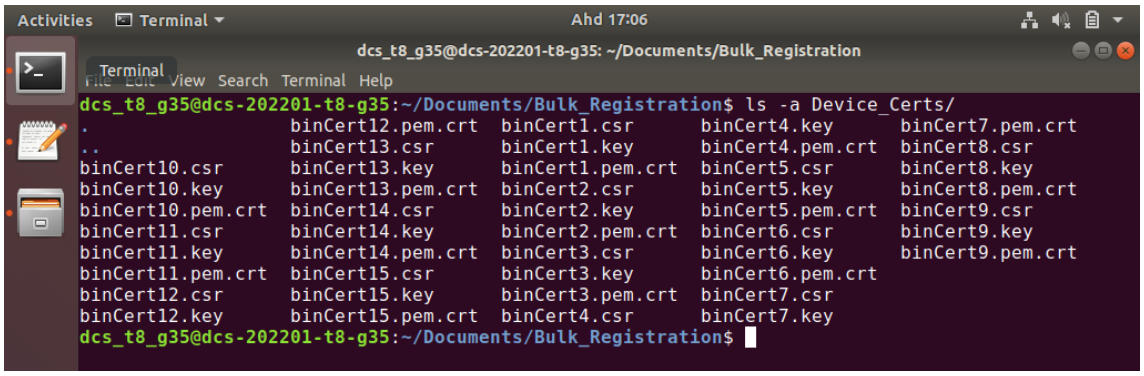


3) Execute the command below to generate the bulk registration parameter values required for the bulk registration. The **i** argument indicates the path to the file containing the device metadata; the **p** argument indicates the directory where the CSR are stored; while the **s** argument indicates the start ID for the device; the **o** argument indicates the output file name. The bash script retrieves the device metadata from the file and write it to a JSON object as shown in the image below. The bash script also uses AWS CLI to request AWS to create device certificate based on the CSR. The certificate ID is then retrieved from the AWS CLI response and written to the same JSON object to associate the newly created certificate with the thing.

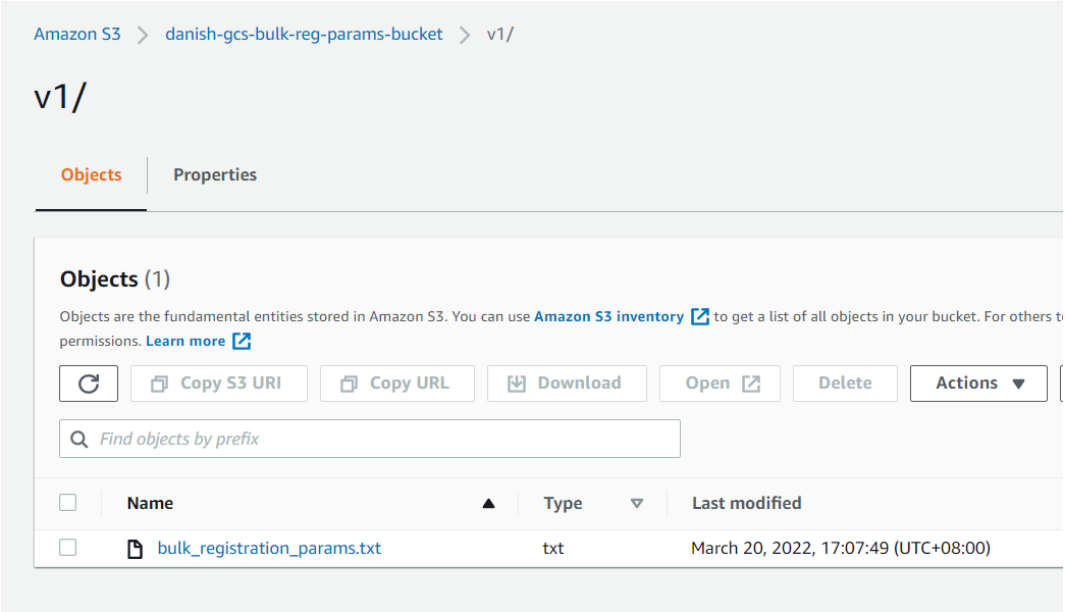
```
./create_bulk_registraton_params.sh -i devices.txt -p Device_Certs -s 1 -o bulk_registration_params.txt
```



4) The **Device_Certs** directory now contains the private key, CSR, and the device certificate.



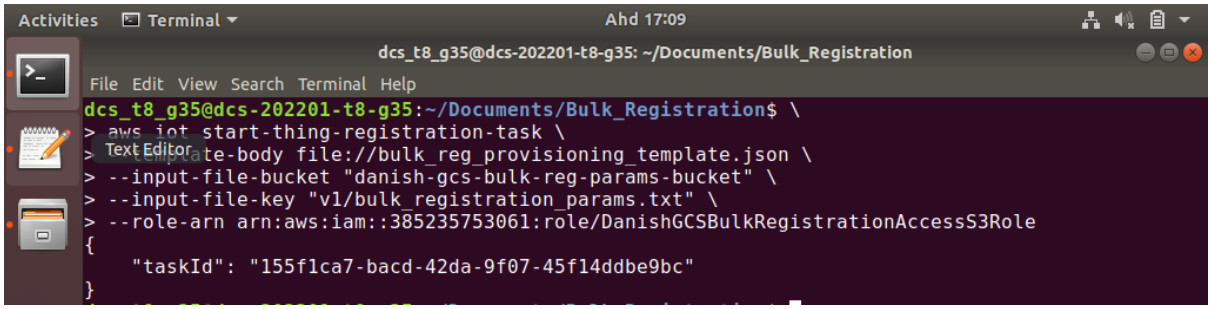
5) Upload the file **bulk_registraton_params.txt** to the S3 bucket.



6) Ensure that the IAM service role named **DanishGCSBulkRegistrationAccessS3Role** is created.

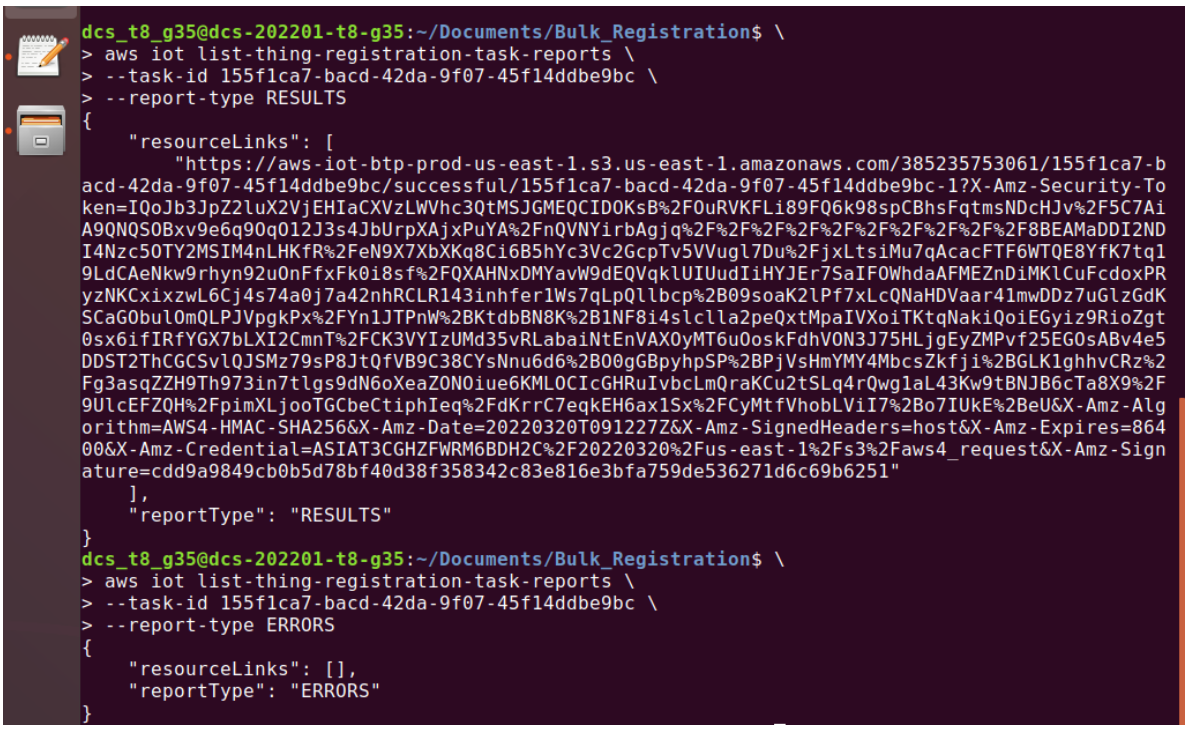
7) Execute the command below to create and start a bulk thing provisioning task.

```
aws iot start-thing-registration-task \
--template-body file://bulk_reg_provisioning_template.json \
--input-file-bucket "danish-gcs-bulk-reg-params-bucket" \
--input-file-key "v1/bulk_registration_params.txt" \
--role-arn
arn:aws:iam::385235753061:role/DanishGCSBulkRegistrationAccessS3Role
```



8) Execute both commands below to check the status of the bulk thing provisioning task for a particular **taskId**. The **resourceLinks** for the report type **ERRORS** will be empty if the task is successful. Access to the resource link to read the logs and retrieve the device certificates created by the AWS, if needed.

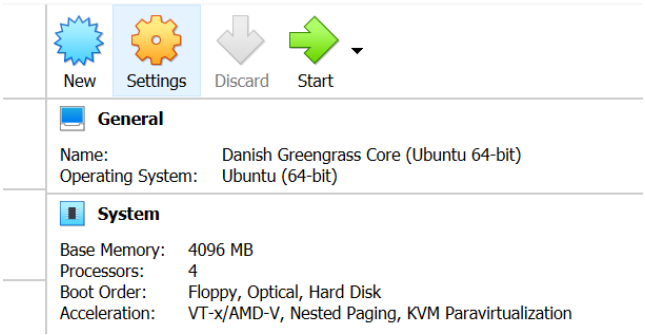
```
aws iot list-thing-registration-task-reports \
--task-id 155f1ca7-bacd-42da-9f07-45f14ddbe9bc \
--report-type RESULTS
aws iot list-thing-registration-task-reports \
--task-id 155f1ca7-bacd-42da-9f07-45f14ddbe9bc \
--report-type ERRORS
```



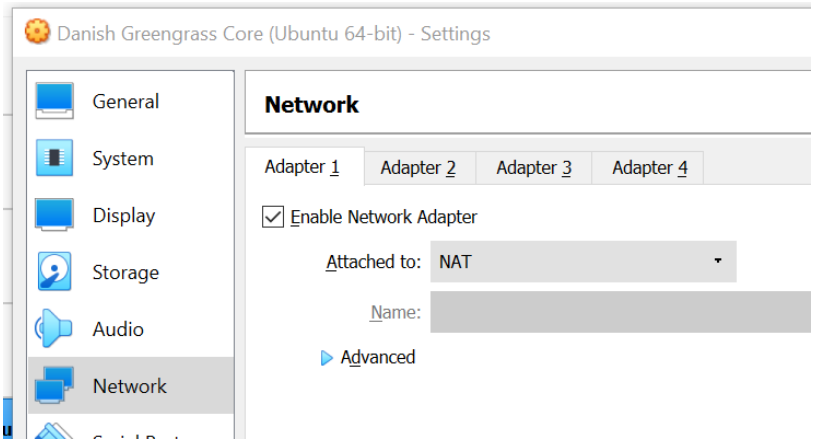
3.2 Establish local connection between client devices and core devices

To simulate LAN environment between two VirtualBox virtual machines,

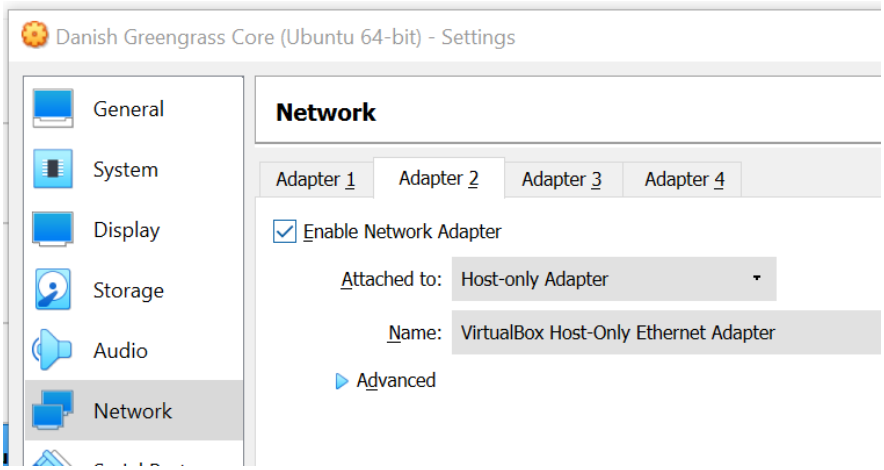
1) Open VirtualBox. Access the settings of the VirtualBox and select **Network**.



2) Ensure that the **Adapter 1** is attached to **NAT** to provide internet connection to the virtual machines. Virtual machines cannot communicate with each other using this adapter.

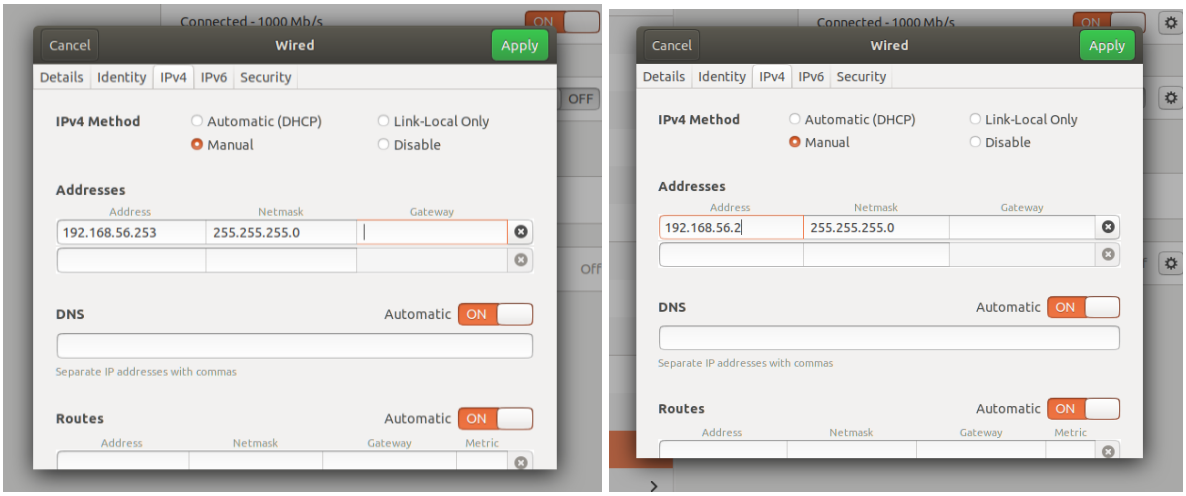


3) **Enable** and attach the **Adapter 2** to **Host-only Adapter** to allow communication between virtual machines as if they were in the same LAN (Oracle Corporation, 2022).



4) Start the virtual machines for both client device and core device. For Ubuntu 18.04, a new network interface named **ensp0s8** is shown like the one in the image below.

5) Access to the network settings to assign static IP addresses to both devices. Assign the client device the IPv4 address **192.168.56.2/24** while assign the core device the IPv4 address **192.168.56.253/24**. Note that the **192.168.56.1/24** has been reserved for host OS. Leave the Gateway address empty since there is no router in this LAN.

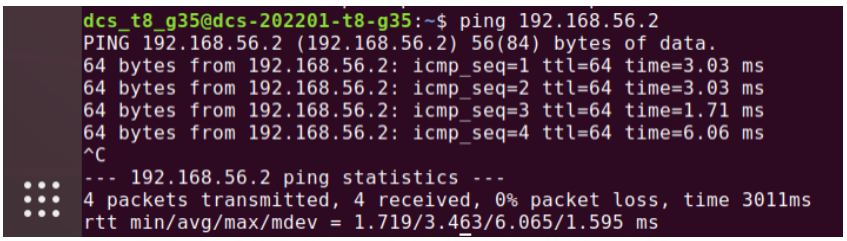


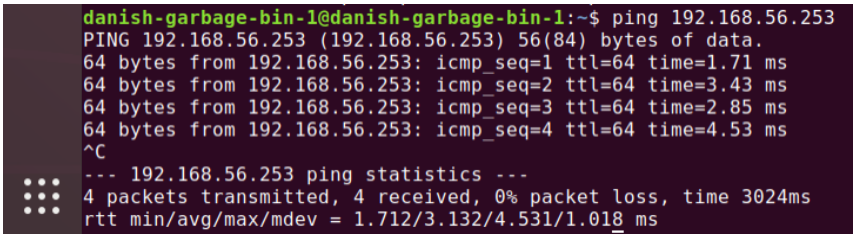
6) It is important to check that the **VirtualBox Host-Only Ethernet Adapter**'s network address is **192.168.56.0/24**. For example, if the host OS is running on Windows, execute **ipconfig/all** and locate the ethernet adapter as shown in the image below.

```
Ethernet adapter Ethernet 3:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : VirtualBox Host-Only Ethernet Adapter
   Physical Address. . . . . . . . . : 0A-00-27-00-00-14
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::3433:92d0:cffd:e199%20(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.56.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 50987047
```

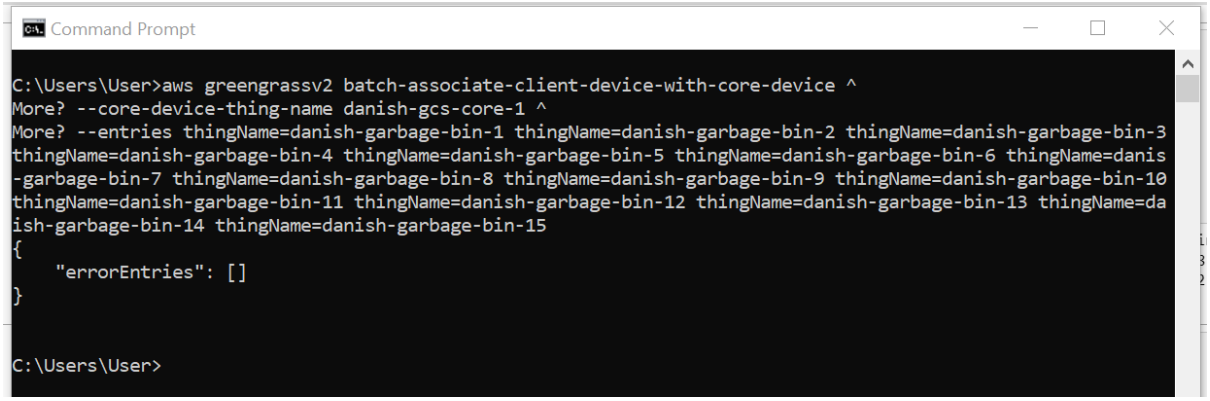
7) Ping each other to check the network configuration.





8) Execute the command below to associate the client devices with the core device. The association allows the client devices to use Greengrass cloud discovery to retrieve associated core devices' connectivity information and certificates (Amazon Web Services, Inc., 2018). In Layman's terms, Greengrass cloud discovery acts like a "local" DNS server to inform the client devices on which local IP address and port number to connect to the core device.

```
aws greengrassv2 batch-associate-client-device-with-core-device ^
--core-device-thing-name danish-gcs-core-1 ^
--entries   thingName=danish-garbage-bin-1   thingName=danish-garbage-bin-2
thingName=danish-garbage-bin-3               thingName=danish-garbage-bin-4
thingName=danish-garbage-bin-5               thingName=danish-garbage-bin-6
thingName=danish-garbage-bin-7               thingName=danish-garbage-bin-8
thingName=danish-garbage-bin-9             thingName=danish-garbage-bin-10
thingName=danish-garbage-bin-11            thingName=danish-garbage-bin-12
thingName=danish-garbage-bin-13            thingName=danish-garbage-bin-14
thingName=danish-garbage-bin-15
```



9) Run the command below to list all the possible endpoints and ports that the client devices can connect to the MQTT broker in the core device. As shown in the image below, the host

address of **192.168.56.253** and port number of **8883** is automatically updated by the IP Detector component that is installed in the core device. Note that **10.0.2.15** is the IP address of the first adapter that only provides internet connection to that virtual machine itself. The network setup is complete as the client device and core device can communicate with each other on **192.168.56.0/24**.

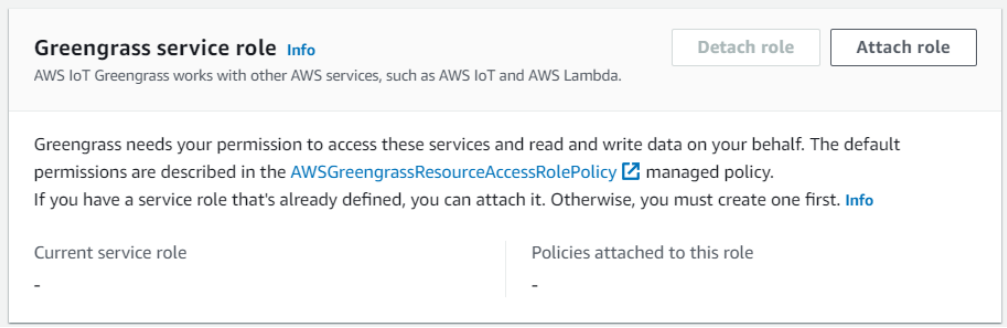
```
aws greengrassv2 get-connectivity-info –thing-name danish-gcs-core-1
```

Section 4: Greengrass Core Device Setup

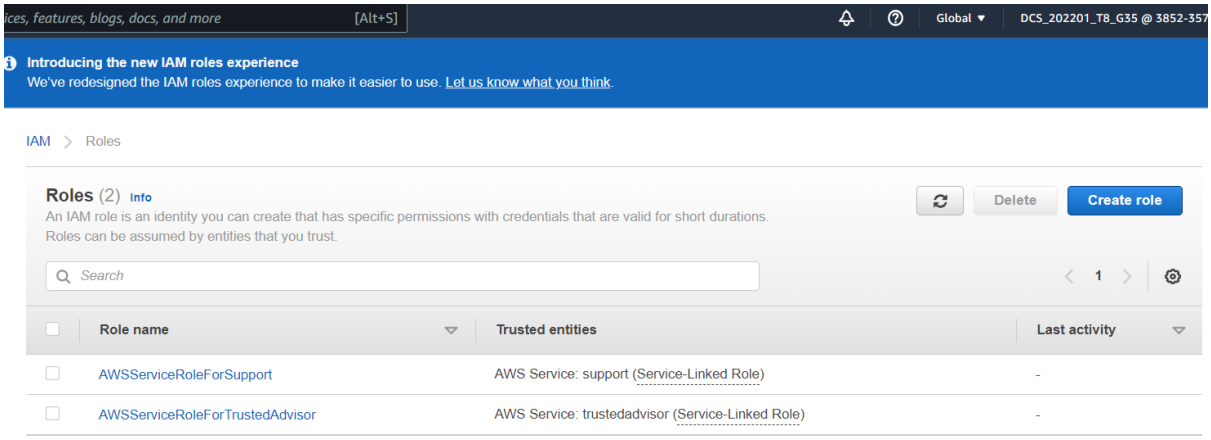
4.1 Create a Greengrass Service Role

1) The Greengrass service role is an AWS IAM service role that authorizes AWS IoT Greengrass to access AWS services and resources. AWS IoT Greengrass assumes the service role to have full permission on all AWS IoT Greengrass services and other related services such as AWS Lambda and Amazon S3. Follow the steps below to create the Greengrass service role.

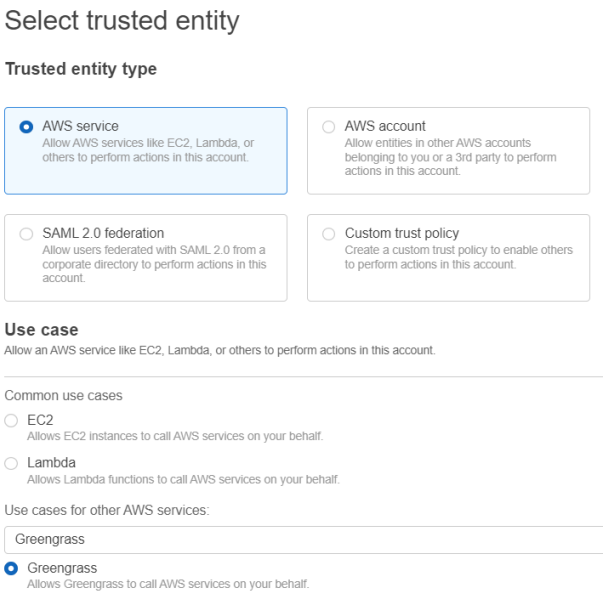
2) Check if the Greengrass service role has already existed in your AWS account. In the **AWS IoT Console** navigation menu, choose **Settings**. In the **Greengrass service role** section, if no service role is seen like the one in the image below, proceed with the rest of the section to create a service role.



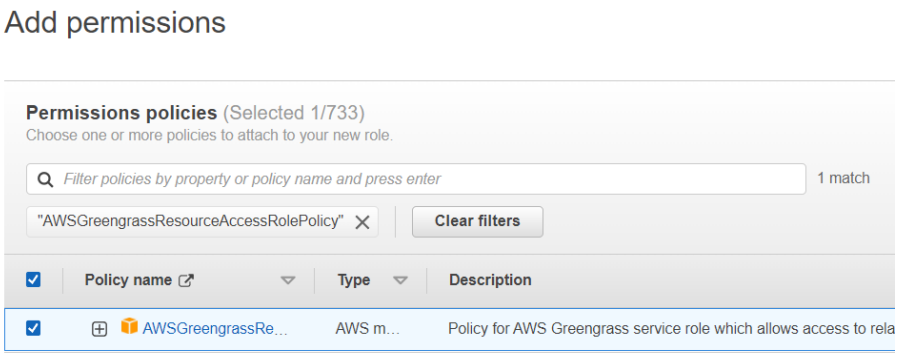
3) In the **IAM console** navigation menu, choose **Roles**, then choose the **Create role** button.



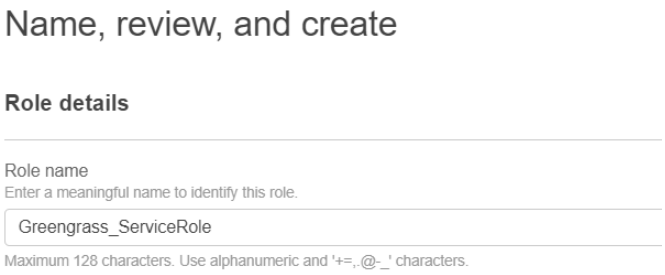
4) Select **AWS service** under **Trusted entity type**. Select **Greengrass** under **Use cases for other AWS services**. Select the **Greengrass** option in the radio buttons. Click **Next**.



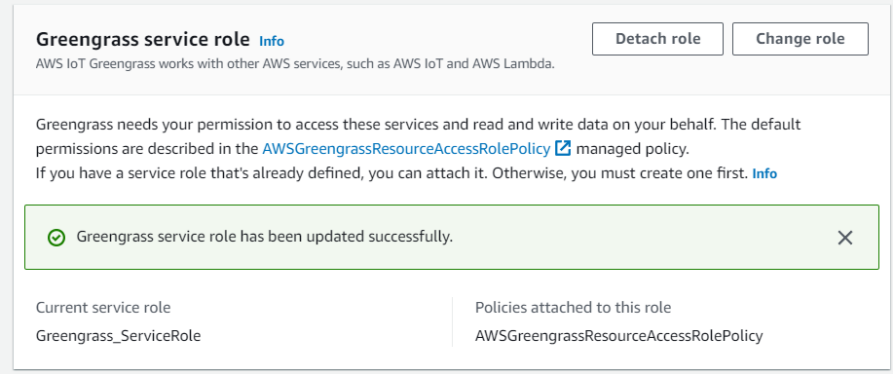
5) Under **Permissions policies**, select the **AWSGreengrassResourceAccessRolePolicy** to attach to the role. Click **Next**.



6) Name the role as **Greengrass_ServiceRole**. Review the role details before click **Create role**.



7) In the **AWS IoT Console** navigation menu, choose **Settings**. In the **Greengrass service role** section, click **Attach role** and select the IAM role that you created.



4.2 Check Device Requirements

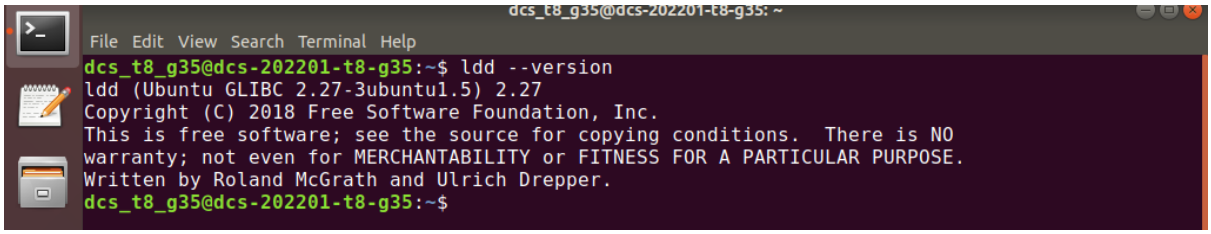
Before proceeding to installing the Greengrass Core Software in the edge device, ensure that the device meets the following requirements:

a) Linux system uses GNU C Library (glibc) version 2.25 or greater.
b) Java Runtime Environment (JRE) version 8 or greater is installed.
c) User must have the permission to run sudo with any user and any group.
d) The /tmp directory must be mounted with exec permissions.
e) Certain shell commands must be available.

4.2.1 Linux system uses GNU C Library (glibc) version 2.25 or greater

1) Ensure that the Linux system uses GNU C Library (glibc) version 2.25 or greater by executing the command below.

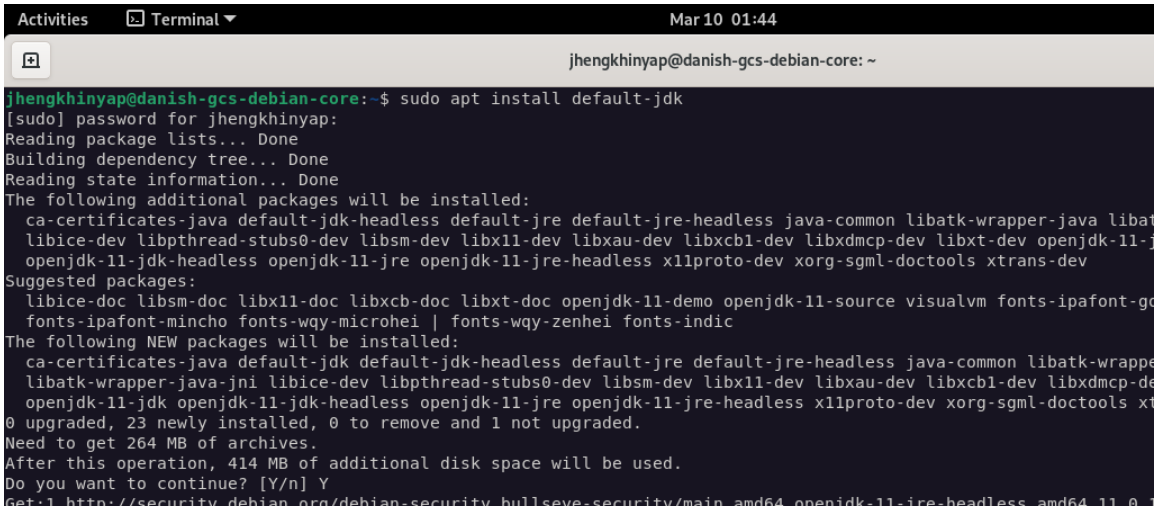
```
ldd --version
```



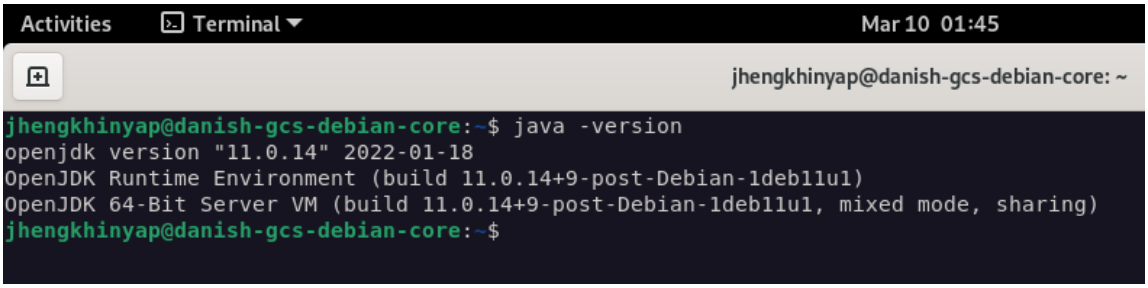
4.2.2 Java Runtime Environment (JRE) version 8 or greater is installed.

1) Execute the command below to install the Java runtime. AWS IoT Greengrass Core software is dependent on Java runtime to execute.

```
sudo apt install default-jdk
java –version
```



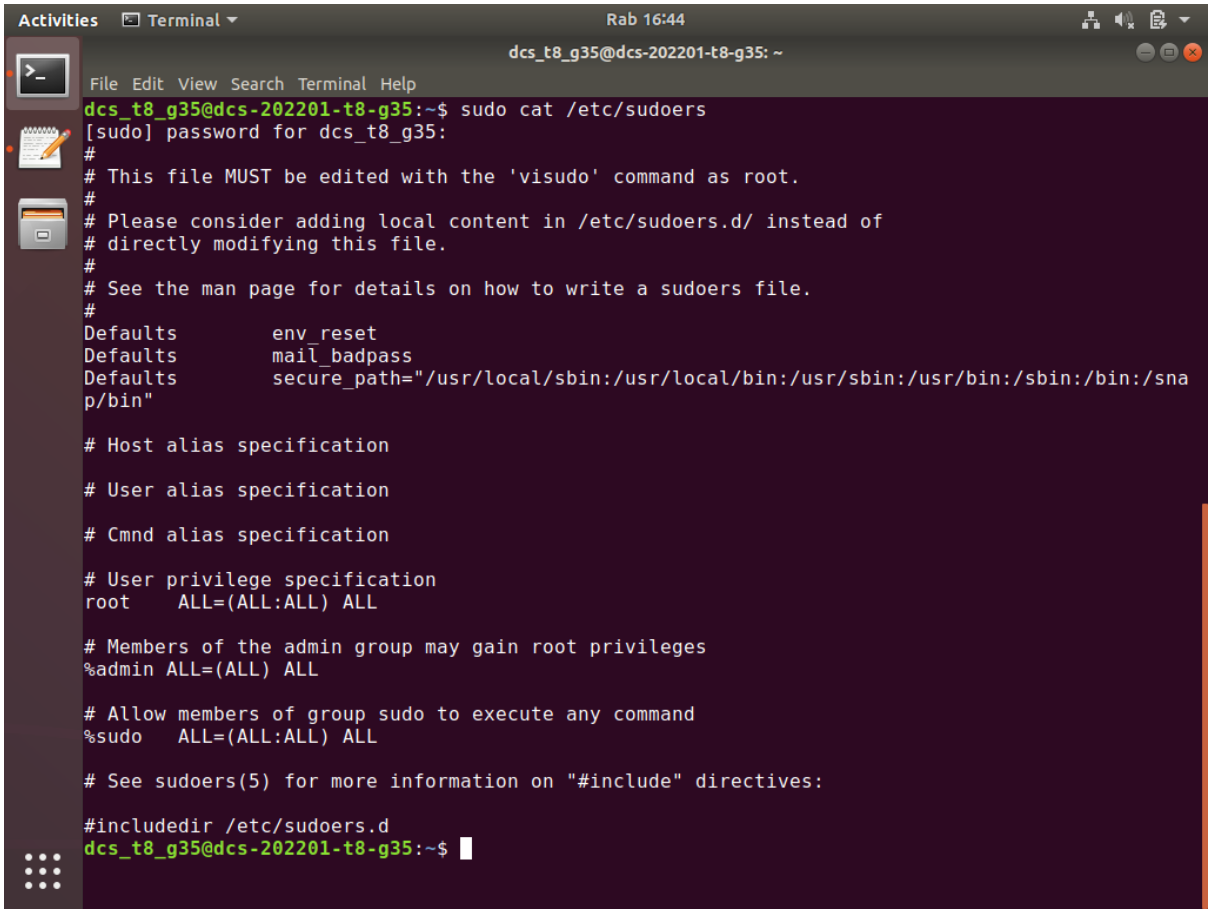
2) Execute the command below to ensure that the installed JDK version is 8 or greater.



4.2.3 User must have the permission to run sudo with any user and any group

1) Verify that the user that runs the AWS IoT Greengrass Core software has the permission to run **sudo** with any user and any group. By default, Ubuntu has enabled this by default. First, run the command below.

```
sudo cat /etc/sudoers
```

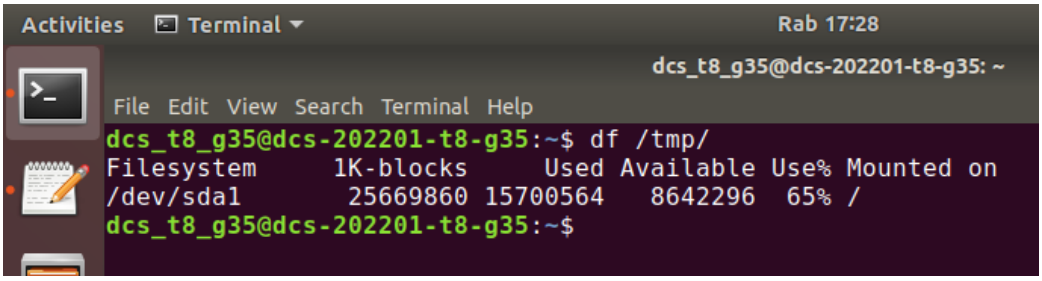


2) In the image above, the line `%sudo ALL=(ALL:ALL)` allows a member of the sudo group to run any command as any user (`sudo -u anyuser`) or any group (`sudo -g anyuser`) (Andrew B, 2013).

3) Edit the /etc/sudoers if the permission is not yet been added.

4.2.4 The /tmp directory must be mounted with exec permissions.

1) In Debian-based Linux distribution such as Ubuntu, /tmp is by default part of the filesystem that is mounted on the root filesystem "/" as shown in the image below.



2) tmpfs is not mounted on /tmp like other Linux distributions, hence the similar record in the image below is not found when executing the command above.

```
Filesystem            Size  Used Avail Use% Mounted on
tmpfs                 256M  688K  256M   1%  /tmp
```

3) Execute the first command below to mount **tmpfs** on /tmp. The command below will create a **tmpfs** filesystem where files stored in /tmp is in virtual memory and thus has faster file access (Kerrisk, M., 2016). The second command is useful when the **tmpfs** is already mounted on /tmp but did not have the permission to execute binaries.

```
sudo mount -o mode=1777,nosuid,nodev -t tmpfs tmpfs /tmp
sudo mount -o remount,exec /tmp
```

4) Execute the command below to check that the mounting is successful. It can be shown in the image that the mounting is successful because of the last two records.

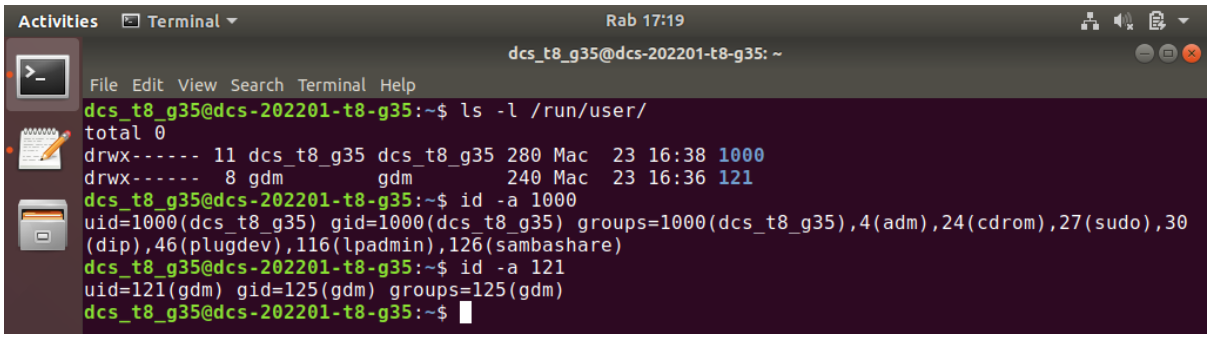
5) There is some prerequisite to understand the output above:

a) If **/proc/mounts** doesn't list **noexec**, then the file system is mounted with the **exec** permission.

b) Since Ubuntu 15.04, systemd is used by default. As a result, the new location of /tmp is in /run/user/$uid. In this virtual machine, /tmp is in /run/user/121 and /run/user/1000. These directories store temporary files for running processes like the Greengrass Core daemon. The reason that each user has a separate and isolated local /tmp is to reduce complications related to simultaneous modification by multiple users (Hoffman, C., 2015; Jain, S., 2019; phemmer, 2014).

c) Note that the uid of the user can be checked by executing the command below.

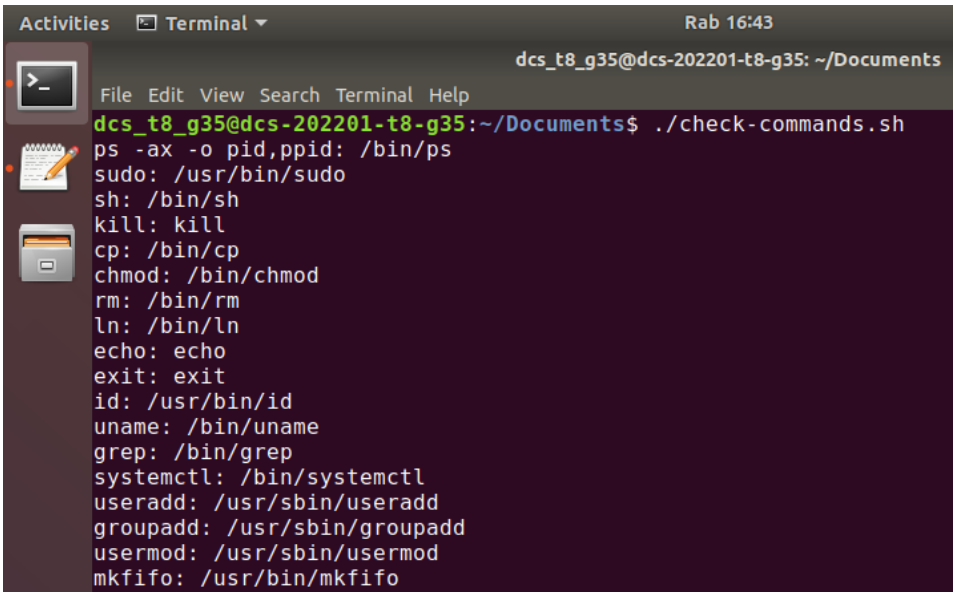
```
ls -l /run/user/
id -a 1000
id -a 121
```



4.2.5 Certain shell commands must be available.

1) Execute the bash script to check that the shell commands are indeed available. The *check-commands.sh* can be found in *core_device/check-commands.sh*.

```
./check-commands.sh
```



4.3 Install Greengrass Core Software with Manual Provisioning

1) Execute the command below to request AWS to generate a 2048-bit RSA key pair and an AWS-signed device certificate. Save the **certificateId** field value from the command's output for later uses.

```
aws iot create-keys-and-certificate --set-as-active \
--certificate-pem-outfile danish-gcs-core-cert/danish-gcs-core-1.pem.crt \
--public-key-outfile danish-gcs-core-cert/danish-gcs-core-1-public.pem.key
\
--private-key-outfile danish-gcs-core-cert/danish-gcs-core-1.pem.key
```

2) Execute the command below to create the thing, attach the IoT policy to the device certificate, and then attach the device certificate to the thing. Substitute the **<new-certificate-id>** with the **certificateId** value that is copied just now.

```
aws iot create-thing --thing-name "danish-gcs-core-1" \
--thing-type-name "danish-gcs-core" \
--attribute-payload "{\"attributes\": {\"version\":\"v1\",
\"serialNumber\":\"0C25NFD9\"}}"

aws iot attach-policy --policy-name DanishGCSCoreThingPolicy --target
arn:aws:iot:us-east-1:385235753061:cert/<new-certificate-id>

aws iot attach-thing-principal --thing-name danish-gcs-core-1 \
--principal arn:aws:iot:us-east-1:385235753061:cert/<new-certificate-id>
```

3) Execute the command below to download and unzip the AWS IoT Greengrass Core software into a folder named **GreengrassInstaller**.

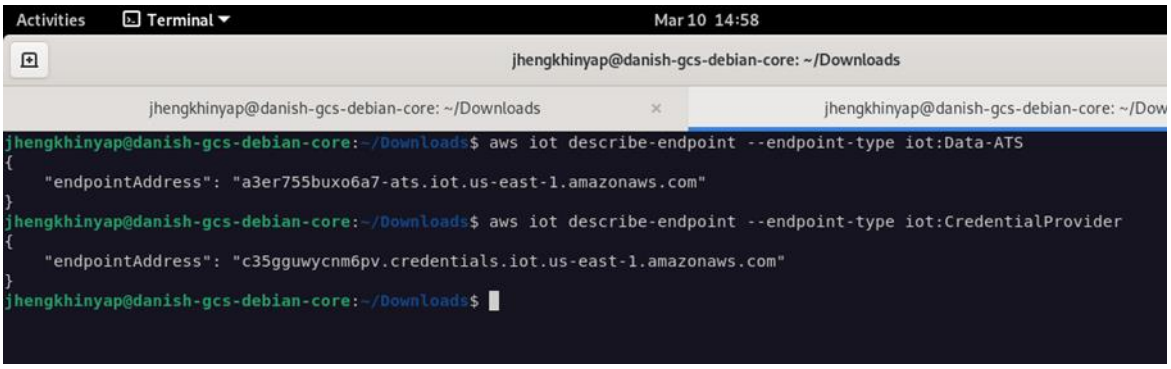
```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm
greengrass-nucleus-latest.zip
java -jar ./GreengrassInstaller/lib/Greengrass.jar –version
```

4) Setup an installation folder named **/greengrass/v2** at the root directory. Copy the private key and device certificate to the folder. Then, download the Amazon Root certificate to the folder.

```
cd ~
cp -r ~/Documents/danish-gcs-core-cert ~
sudo mkdir -p /greengrass/v2
sudo chmod 755 /greengrass
sudo cp -r danish-gcs-core-cert/* /greengrass/v2
sudo curl -o /greengrass/v2/AmazonRootCA1.pem
https://www.amazontrust.com/repository/AmazonRootCA1.pem
ls /greengrass/v2/
```

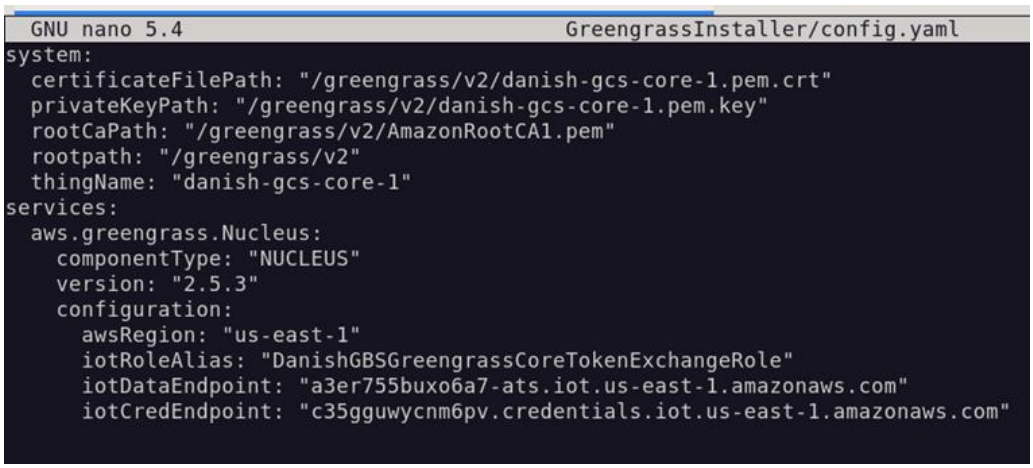
5) Execute the commands below to retrieve the ATS signed data endpoint and IoT credentials provider API endpoint, respectively.

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```



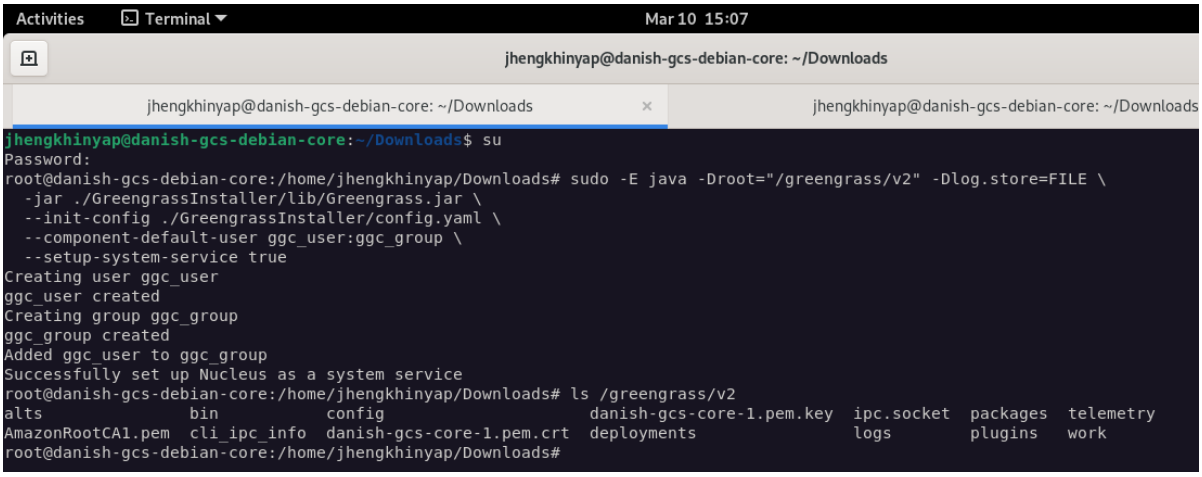
6) Execute the command below to edit the **config.yaml** in the **GreengrassInstaller** directory. Replace the values of iotDataEnpoint and iotCredEndpoint with the previous command outputs accordingly. The content of the config.yaml can be found in *core_device*/*config.yaml.*

```
nano GreengrassInstaller/config.yaml
```



7) Execute the command below to install the Greengrass Core software. Set the **deploy-dev-tools** flag to true to enable local deployment of custom Greengrass component to save development time. The installer will create a user group named **ggc_group** with a user named **ggc_user**. The image below shows the output of a successful installation. If the installation is still not successful after few troubleshooting and fixes, consider the second option in the next section.

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --deploy-dev-tools true
  --setup-system-service true
```



4.4 Install Greengrass Core Software with Automatic Provisioning

1) There is another alternative option to install AWS IoT Greengrass Core software. By setting the **provision** flag to true, the installer will automatically provision the AWS IoT thing, AWS IoT thing group, IAM role, and AWS IoT role alias that the core device requires to operate (Amazon Web Services, Inc., 2022c). This option does not require to edit config.yaml. Set the **deploy-dev-tools** flag to true to enable local deployment of custom Greengrass component to save development time. Execute the command below to install the Greengrass Core software. The full installation instruction can be found at the official AWS Documentation.

```
sudo -E java -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region "us-east-1" \
  --root /greengrass/v2 \
  --thing-name "danish-gcs-core-1" \
  --thing-group-name "danish-gcs-core" \
  --thing-policy-name "DanishGCSCoreThingPolicy"\
  --tes-role-name "DanishGBSGreengrassV2TokenExchangeRole"\
  --tes-role-alias-name "DanishGBSGreengrassCoreTokenExchangeRole" \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --deploy-dev-tools true \
  --setup-system-service true
```

2) Ensure that the installer prints the messages as stated below (the screenshot of installation is lost).

The installer prints the following messages if it succeeds:

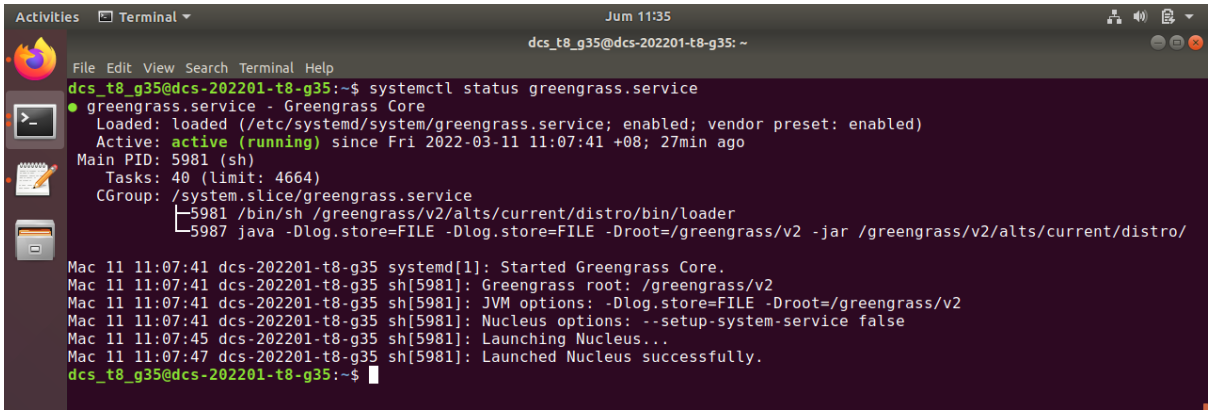
- If you specify `--provision`, the installer prints `Successfully configured Nucleus with provisioned resource details` if it configured the resources successfully.
- If you specify `--deploy-dev-tools`, the installer prints `Configured Nucleus to deploy aws.greengrass.Cli component` if it created the deployment successfully.
- If you specify `--setup-system-service true`, the installer prints `Successfully set up Nucleus as a system service` if it set up and ran the software as a service.

3) This approach is preferable since the first installation option sometimes cannot register the subsequent device if the same thing name is used in the new core device, even if the AWS resources associated to that thing name has already been deleted. Put it simply, use this option to avoid complications.

4.5 Validate Greengrass Core Software Installation

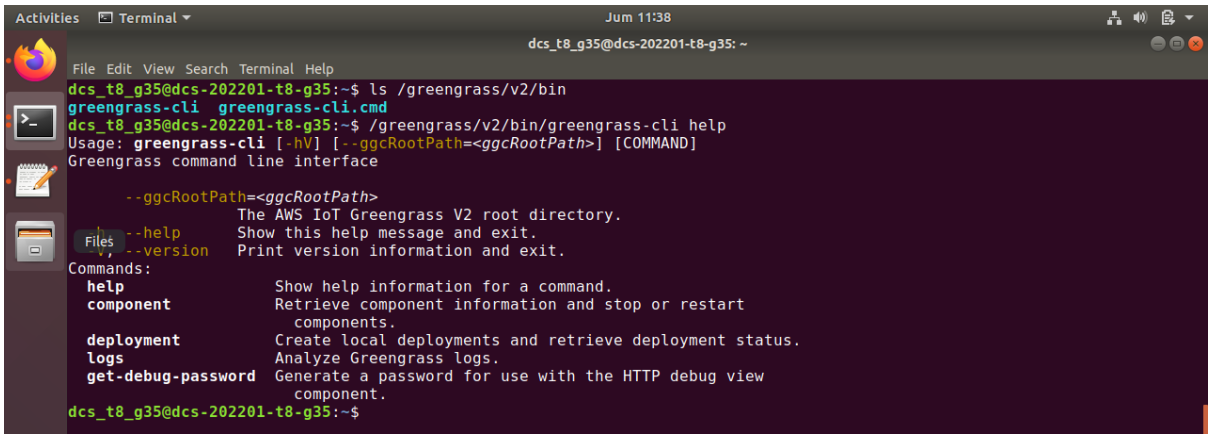
1) Finally, check that the Greengrass Core software has been successfully installed by executing the commands below.

```
systemctl status greengrass.service
```



2) If the **deploy-dev-tools** flag is set to true, execute the command below to ensure that the greengrass-cli has been successfully installed.

```
ls /greengrass/v2/bin
/greengrass/v2/bin/greengrass-cli
```



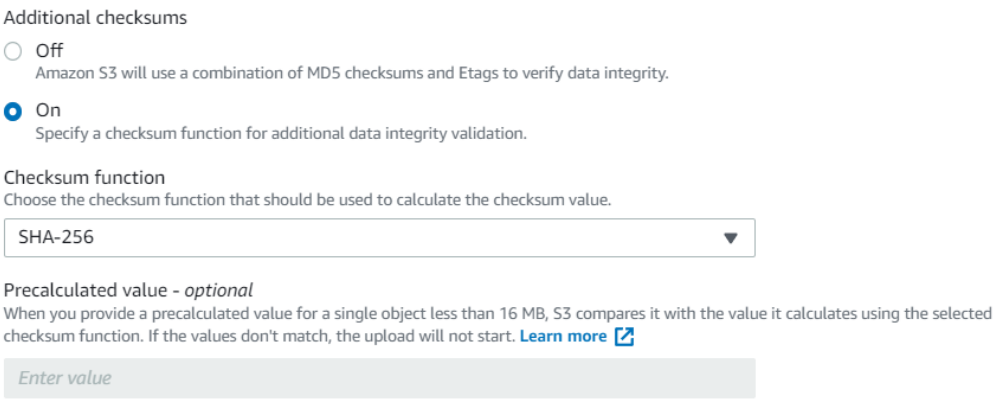
4.6 Prepare Greengrass Component's Artifacts for Cloud Deployment

1) During cloud deployment of Greengrass component, the component's artifacts will be downloaded from the S3 bucket during installation lifecycle. To upload the artifacts to the cloud, in the **Amazon S3 Console** navigation menu, select **Buckets > danish-gcs-model-artifacts-bucket**. Create the folders and upload the files by following the file structure shown below. The artifacts can be found in *s3_artifacts* folder.

2) The directory of S3 bucket is visualized as shown below:

danish-gcs-model-artifacts-bucket/v1
> aws.greengrass.DLRImageClassification.artifacts
> trash-cf-inference.zip
> danish.GCS.TelemetryIngestion
> data-ingestion-artifacts.zip
> neo-compiled
> danish_trash_cf_model-LINUX_X86_64.zip
> variant.DLR.artifacts
> environment.yaml
> model-dependencies-installer.zip

3) Before selecting the **Upload** button, select **Properties**. Under the **Additional checksums** section, turn on the additional checksum and select **SHA-256** in the **Checksum function** drop-down menu. Leave the **Precalculated value** field empty.



4) After uploading the file, the generated checksum value can be found by selecting the uploaded object in the data table. The value is then copied into the **Manifests** > **Artifacts** > **Digest** field in the component recipe. The checksum is important for the Greengrass Core component installer to check for whether a file is corrupted due to transmission error or poor network connectivity.

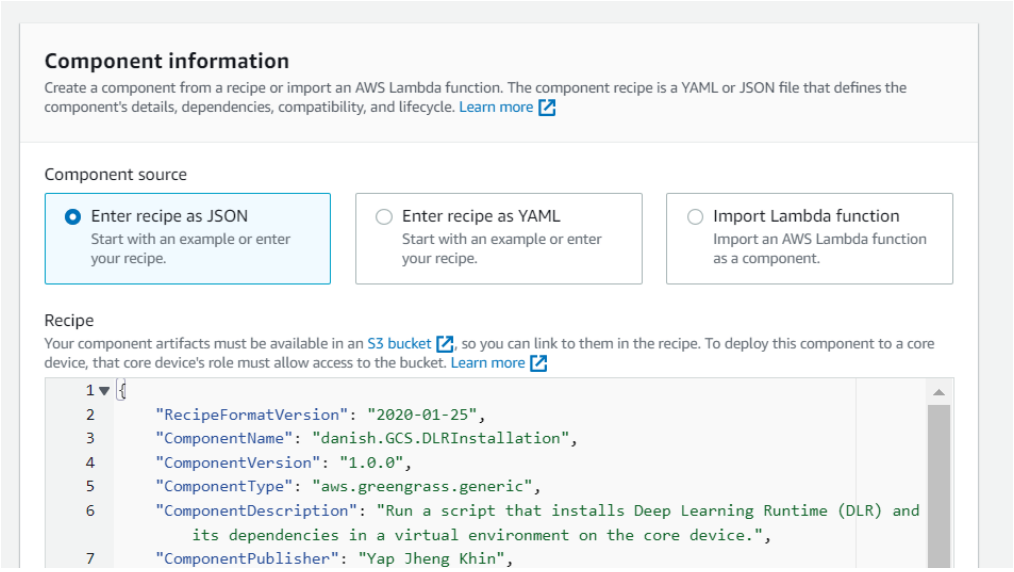
4.7 Deploy AWS IoT Greengrass Component

(Make sure all the artifacts had been uploaded to the S3 bucket before continuing this section)

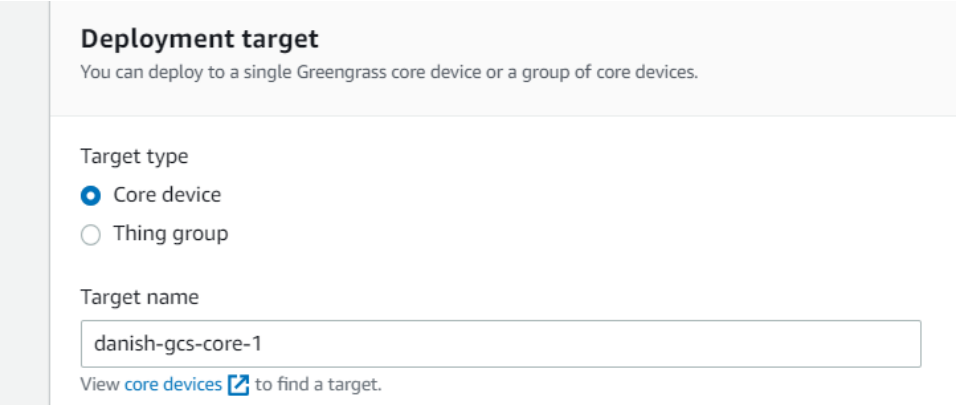
1) In the **AWS IoT Console** navigation menu, select **Greengrass** > **Components**. Select **Create component** under **My components** section.

2) Select **Enter recipe as JSON** as the **Component source**. Copy the JSON file contents from the *component_recipes* directory before selecting **Create component**. Create the components in the following order.

i. danish.GCS.DLRInstallation
ii. danish.GCS.TrashClassification.ModelStore
iii. danish.GCS.TelemetryIngestion
iv. danish.GCS.TrashClassification

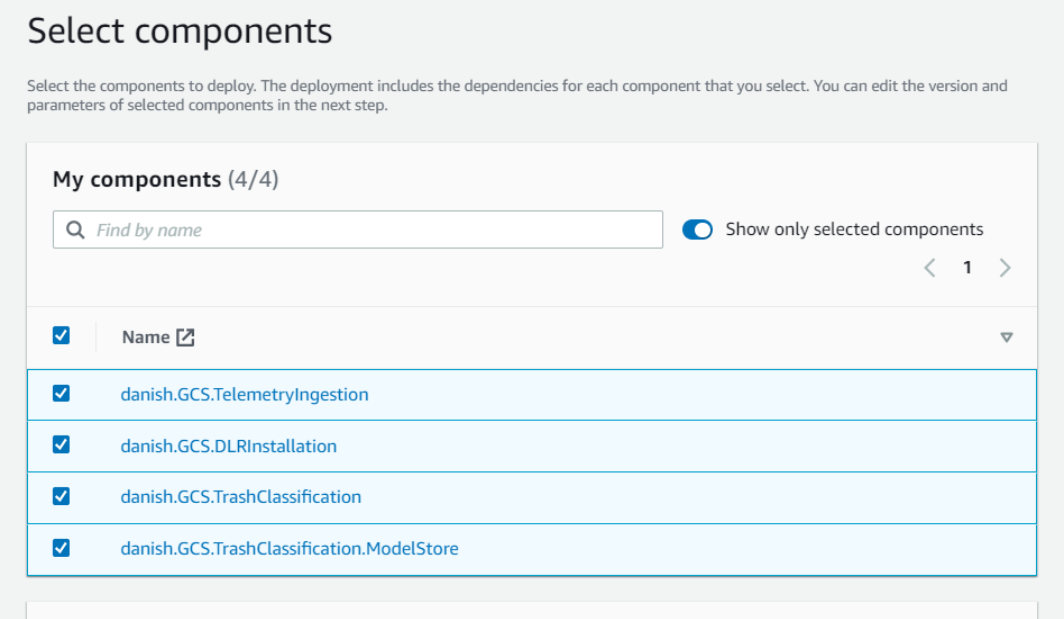


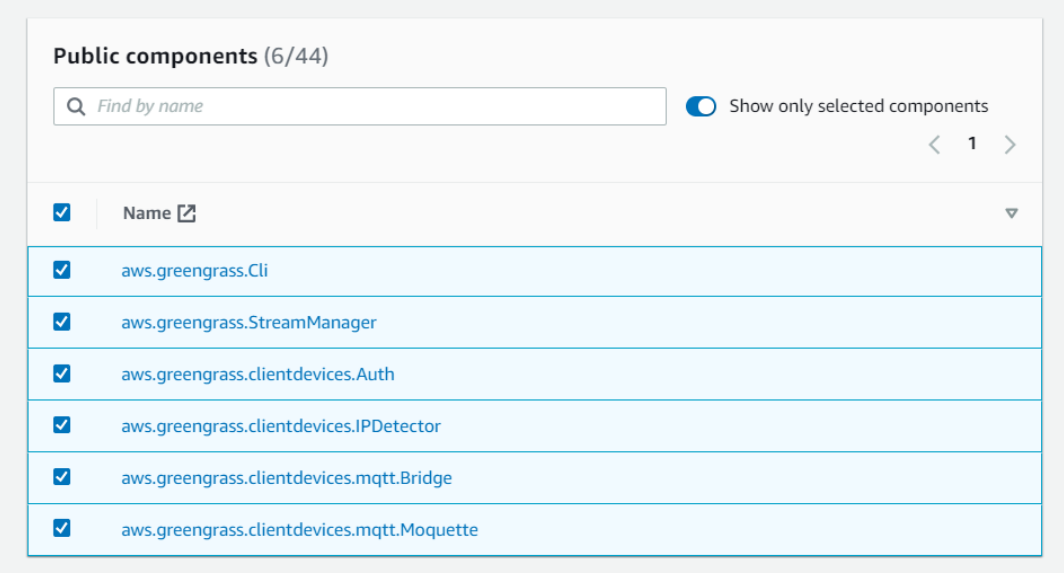
3) In the **AWS IoT Console** navigation menu, select **Greengrass** > **Deployments**. Select **Create** if no deployments are shown.



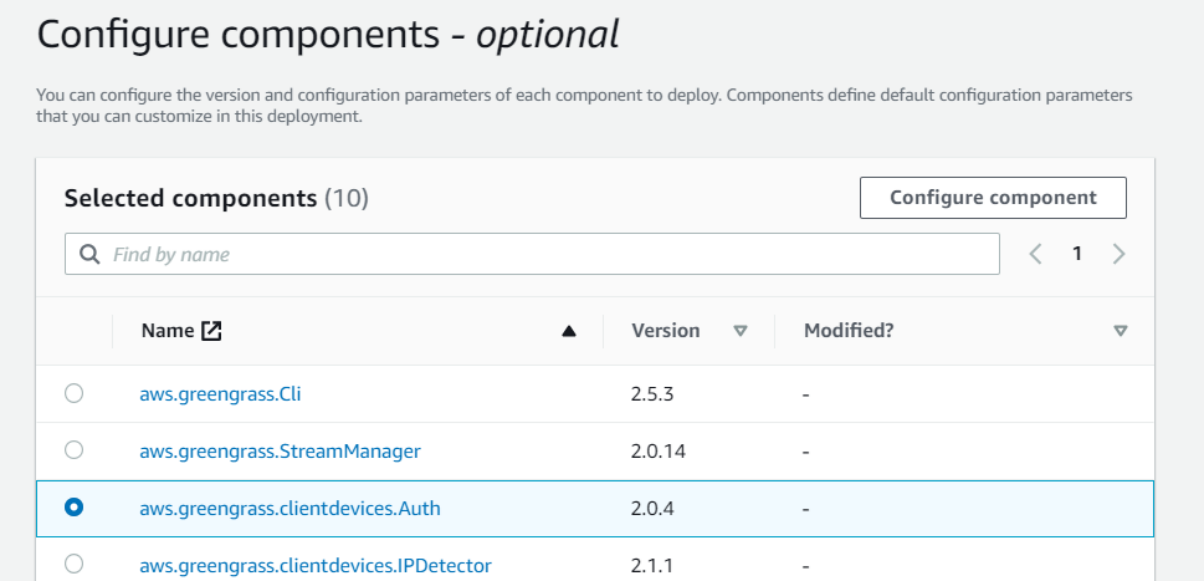
4) In **Step 1: Specify target**, Provide a **name** to the deployment. Select **Core device** as the deployment target and enter **danish-gcs-core-1**. Select **Next**.

5) In **Step 2: Select component**, select the components as demonstrated in the following images below. Select **Next**.



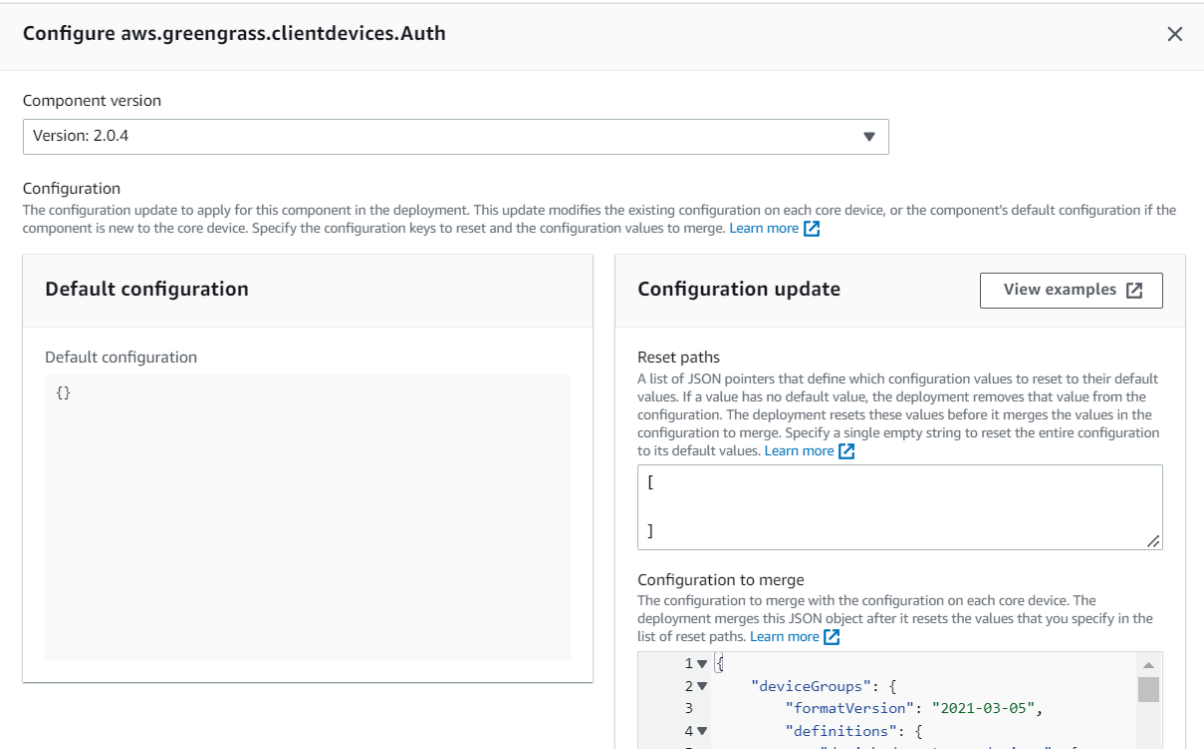


6) In **Step 3: Configure components**, select **aws.greengrass.clientdevices.Auth** > **Configure component**.



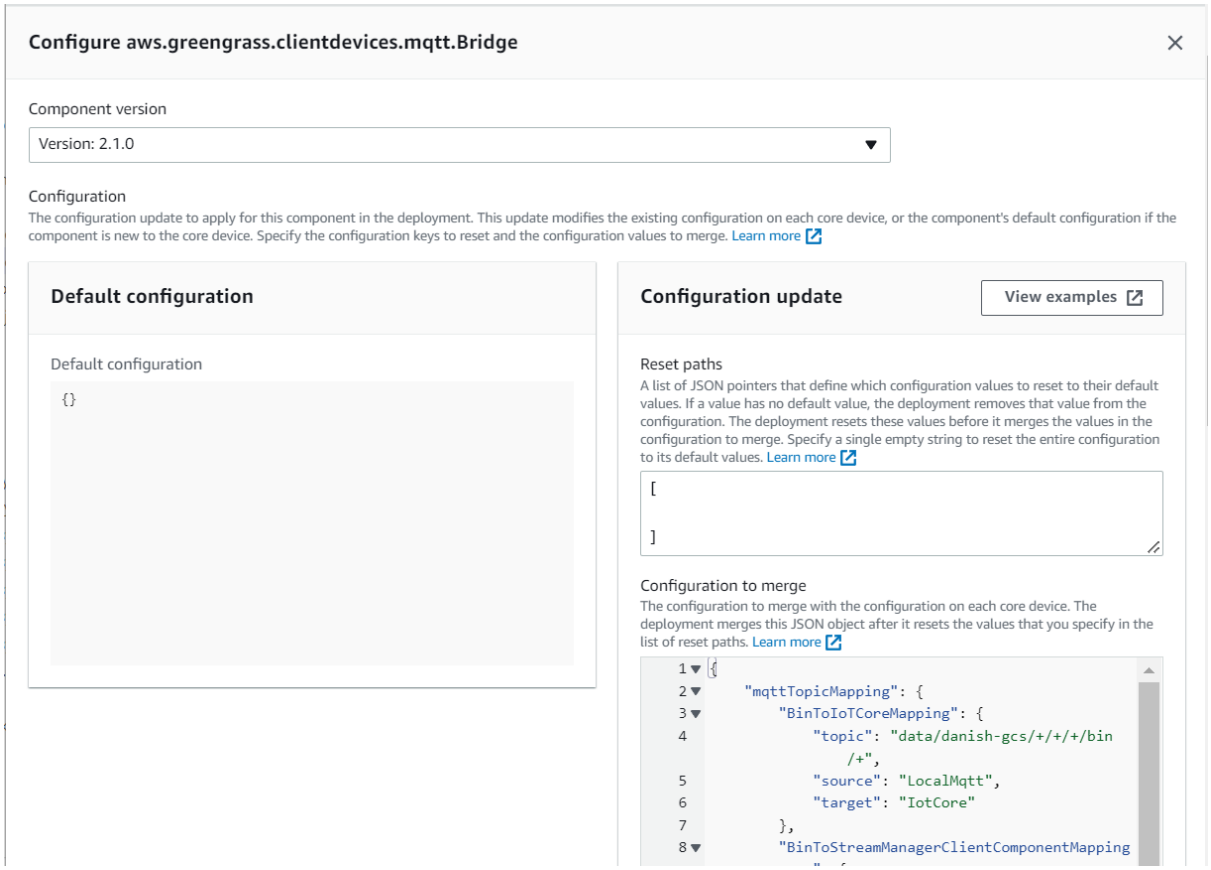
7) Copy the JSON file contents from the *component_recipes/ aws.greengrass.clientdevices.Auth.json* into the bottom-right section. Select **Confirm**.

(The detailed explanation of the component configuration can be found in this section.)



8) Repeat the same for **aws.greengrass.clientdevices.mqtt.Bridge**. The file content is in *component_recipes/aws.greengrass.clientdevices.mqtt.Bridge.json.* Select **Confirm** before selecting **Next**.

(The detailed explanation of the component configuration can be found in this section.)



9) Skip **Step 4: Configure advanced settings**. Review the deployment and select **Deploy**.

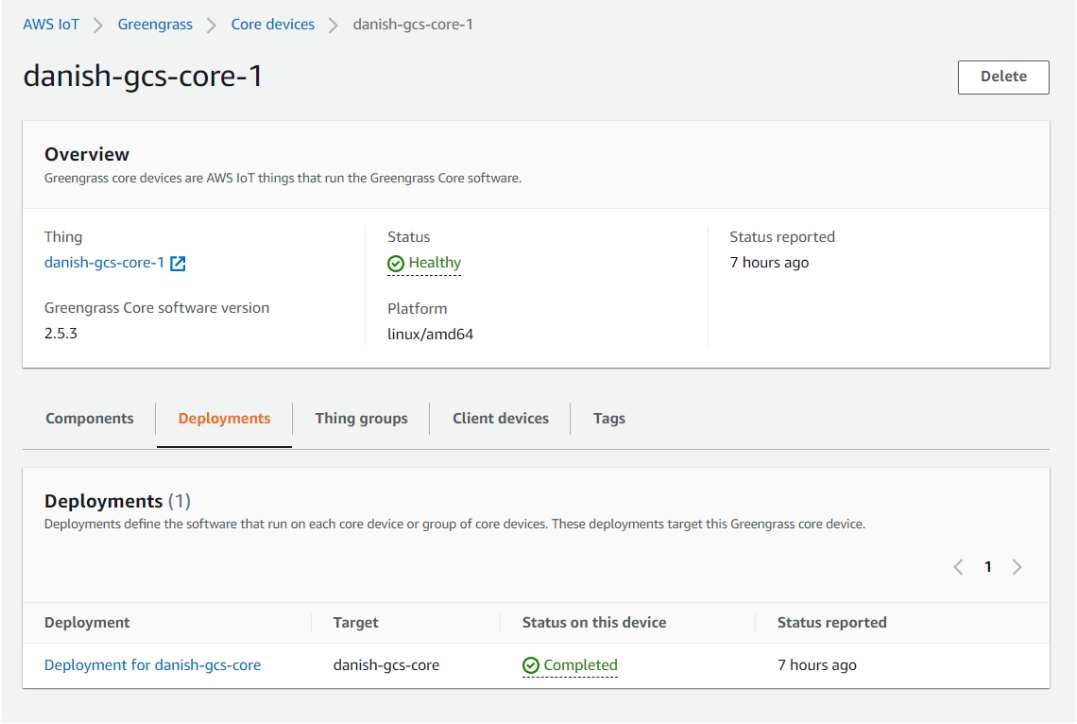
10) To monitor deployment for **dansh-gcs-core-1**, select **Greengrass** > **Core devices** > **danish-gcs-core-1**. First, check whether the device status is **Healthy** or **Unhealthy**. If the device status is **unhealthy**, check the deployment status of the device in the **Deployments** tab. It will transition from **Queued** > **In progress** > **Completed/Failed**. If the deployment status is **Completed**, check if any component is in **Broken** state in the **Components** tab.

11) If the deployment status has **failed**, execute the commands below in the core device to locate the general errors. If any component status is **broken**, replace the log file name to the specific component name like the one shown below.

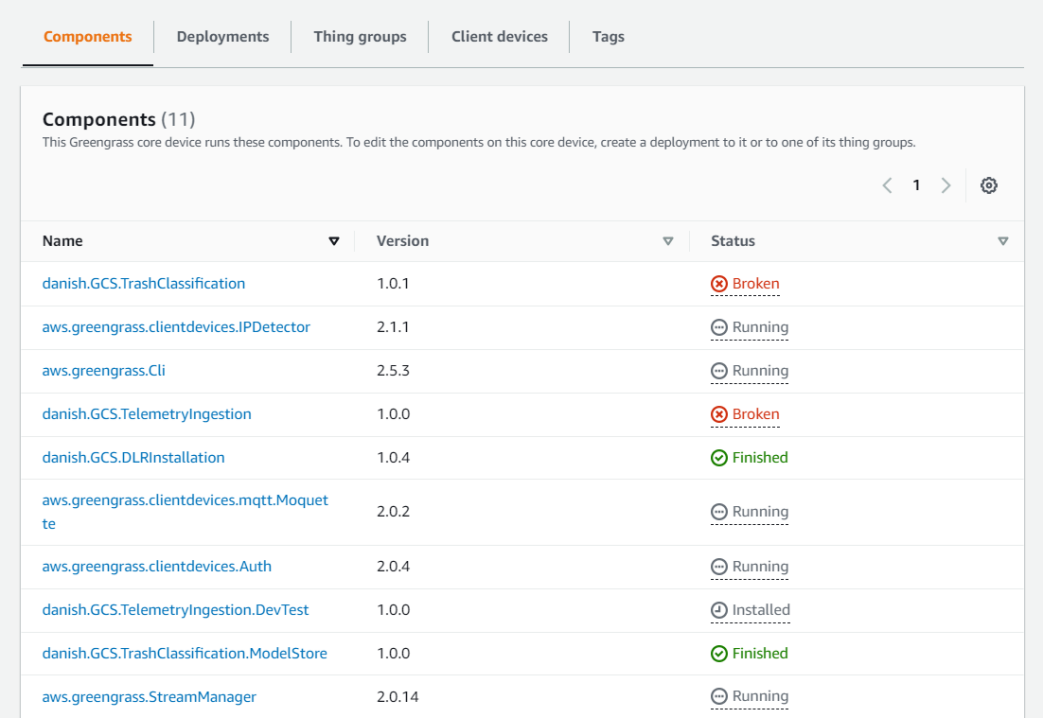
```
sudo cat /greengrass/v2/logs/greengrass.log
```

```
sudo cat /greengrass/v2/danish.GCS.TrashClassification.log
```

12) The example below showed that device status is **healthy** and deployment status is **completed**.



13) The example below showed that the danish.GCS.TrashClassification and danish.GCS.TelemetryIngestion is in **Broken** state. In this assignment, these two components should be in **Running/Installed** state, while danish.GCS.TrashClassification.ModelStore and danish.GCS.DLRInstallation and should be in **Finished** state.



4.8 Data Ingestion Component Validation

1) In the client device, activate the Python virtual environment. Then, run the command below to publish fill level telemetry to the core device and the AWS IoT Core. Refer to the next three subsections to validate the component.

```
python3 ./pub_telemetry_to_core.py \
--thing-name   "danish-garbage-bin-1" \
--dev-cert     "./Device_Certs/binCert1.pem.crt" \
--dev-key      "./Device_Certs/binCert1.key" \
--root-ca      "./CA_Cert/AmazonRootCA1.pem" \
--pub-topics   "./pub_telemetry_topics.txt" \
--message-path "./Telemetry" \
--qos          "at_least_once" \
--verbosity    "info"
```

Argument name	Description
thing-name	The thing name that is used to authenticate with the Moquette MQTT broker.
dev-cert	The device's certificate that is used for authentication.
dev-key	The device's private key that is used for authentication.
root-ca	The root CA or CA certificate to verify the device's certificate. In this assignment, Amazon Root CA 1 is used as verification certificate for the Amazon-signed client devices' certificates.
pub-topics	A file containing a list of topics to publish the telemetry to. The sample file is included in *client_device/pub_telemetry_topics.txt.*
message-path	A directory containing files that each contains telemetry records. The sample directory is included in *client_device/Telemetry.*
qos	Quality of service used in publishing messages. Possible values are "at_most_once" and "at_least_once". "exactly_one"/QOS 2 is not supported by AWS IoT MQTT protocols.
verbosity	The verbosity level used in logging. Possible values are "nologs", "fatal", "error", "warn", "info", "debug", "trace".

4.8.1 Validate Command's Logs from Garbage Bin/Client Device

1) The full sample log file of the command can be found in *logs/pub_telemetry_to_core-Success.log.*

2) The logs indicate that the client device has successfully published the fill level telemetry to the core device and the AWS IoT Core. For this assignment, each fifteen client devices publish five telemetry records. Moquette MQTT broker component expects one client device to own one device certificate. Since only one Ubuntu virtual machine is used as the client device, the **danish-garbage-bin-1** is authorised to publish the telemetry record on behalf of other client devices for testing purposes.

4.8.2 Validate Component's Logs from Core Device

1) The program has logged messages to the **danish.GCS.TelemetryIngestion.log** located in the **/greengrass/v2/logs** directory. The full sample log file of the component can be found in *logs/danish.GCS.TelemetryIngestion-Success.log*.

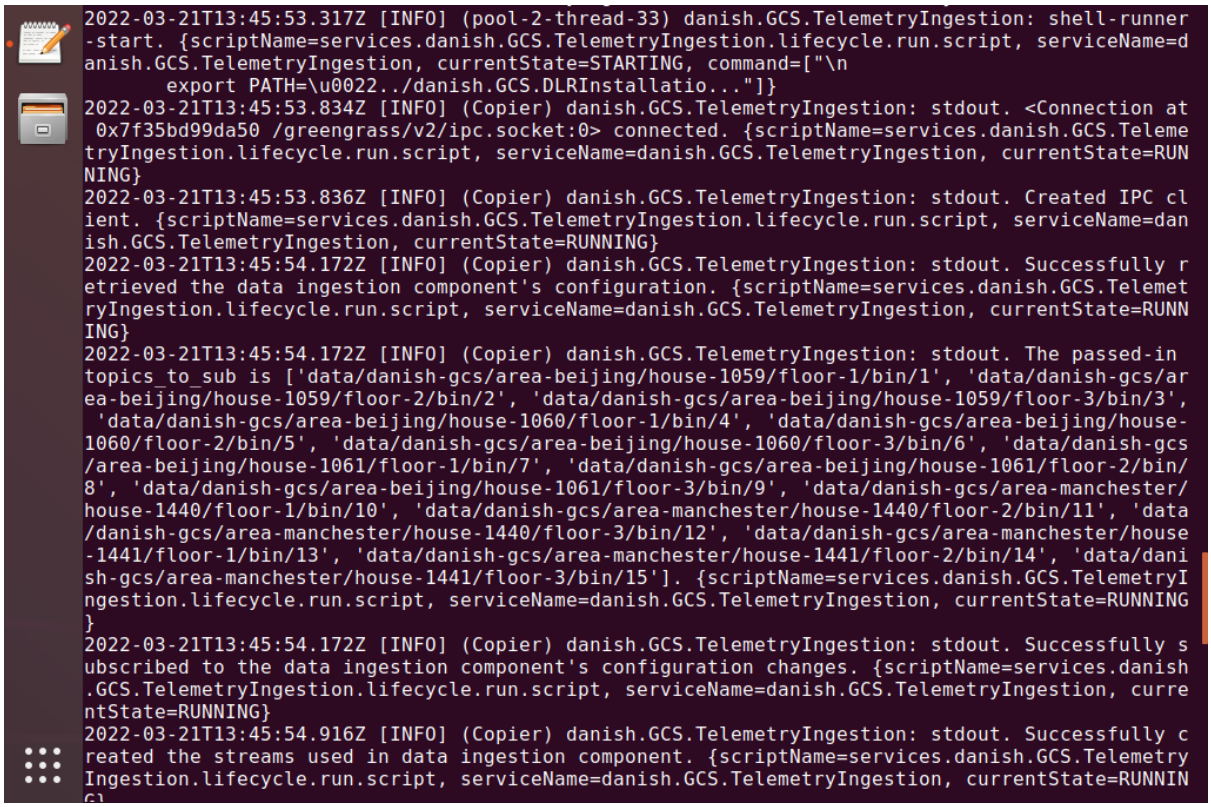
2) Run the command below to view the logs.

```
sudo tail -f /greengrass/v2/logs/danish.GCS.TelemetryIngestion.log
```

3) For the remaining section, the important sections in the log file are discussed to ensure that the readers understand the healthy behaviour of the **TelemetryIngestion** component.

4) To ensure the component has successfully installed and initialized, ensure that the inference component has successfully:

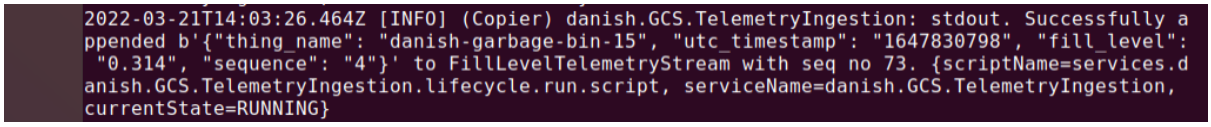
- Created IPC client to subscribe to MQTT topics.
- Subscribed to the component's configuration changes to fetch the latest configuration from the component recipe.
- Created the stream to export fill level telemetry to the AWS IoT Analytics channel.



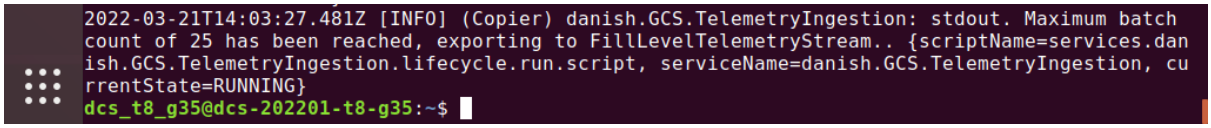
5) The logs below indicate that the component has successfully subscribed to the telemetry topics that the client devices publish to.

6) The logs below indicate that the component has successfully received the telemetry from the client devices.

7) The logs below indicate that the component has successfully appended the telemetry to the stream.



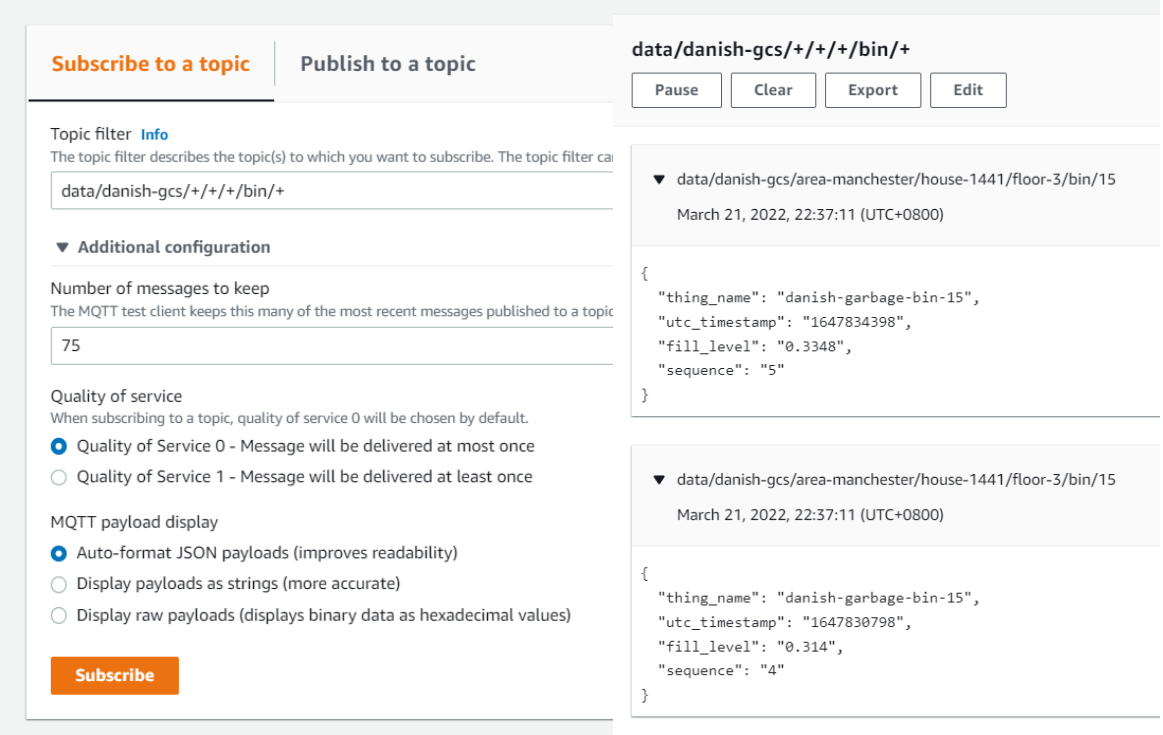
8) The logs below indicate that the stream manager starts to export the telemetry stream to the IoT Analytics channel for every twenty-five telemetry records that are appended to the stream.



4.8.3 Validate Results from AWS Management Console

1) Check that the MQTT bridge component has successfully relay client devices' published messages to the AWS IoT Core. In the **AWS IoT Console** navigation menu, select **Test** > **MQTT test client**. Select **Subscribe to a topic**. Enter the details as shown in the left image below before selecting **Subscribe**. An example of messages received is shown in the right image below.

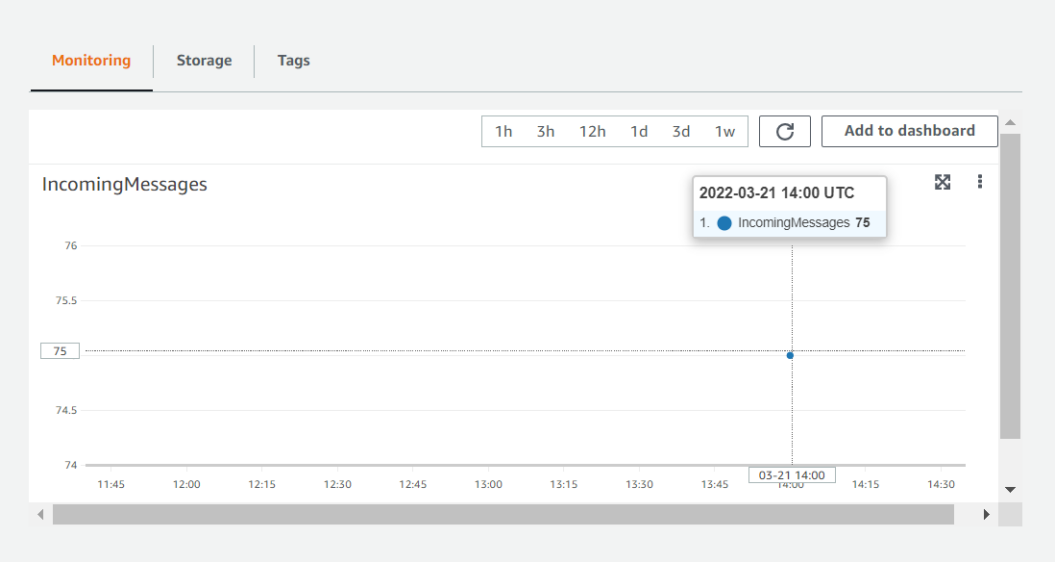
Topic filter: data/danish-gcs/+/+/+/bin/+



2) If the validation above is successful, check that the DynamoDB items has been updated accordingly. In the **DynamoDB Console** navigation menu, select **Tables** > **DanishGarbageBin > Explore table items**. If not, check the CloudWatch logs for any error.

3) Check that the fill level telemetry has exported to the user defined AWS IoT Analytics channel. In the **AWS IoT Analytics Console** navigation menu, select **Channels**. Select the

channel that receives the telemetry. Check the number of incoming messages using the graph under the **Monitoring** tab. Normally, there is around 5 minutes delay between the core device exporting data to the IoT Analytics and the channel showing the updated record in the graph.



4) If the validation above is successful, check that the user defined pipeline has successfully enrich and process the incoming messages before storing in the user defined data store. In the **AWS IoT Analytics Console** navigation menu, select **Pipelines**. Select the pipeline that was created earlier. Check if there is any **ActivityExecutionError** using the graphs under the **Monitoring** tab. Normally, there is around 10 minutes delay between the core device exporting data to the IoT Analytics and the pipeline showing the updated records in the graphs.

4.9 Inference Component Validation

1) In the client device, activate the Python virtual environment. Then, run the command below to request trash classification inference from the core device. Refer to the next three subsections to validate the component.

```
python3 ./pub_sub_to_core_device.py \
--thing-name "danish-garbage-bin-1" \
--dev-cert  ./Device_Certs/binCert1.pem.crt \
--dev-key   ./Device_Certs/binCert1.key \
--root-ca   ./CA_Cert/AmazonRootCA1.pem \
--pub-topic "command/danish-gcs/core/1/trash-classification" \
--sub-topic "command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res" \
--mode      "both" \
--seq-no    1 \
--qos       "at_least_once" \
--sub-count 12 \
--messages ./inference_req_payload.json \
--verbosity "info"
```

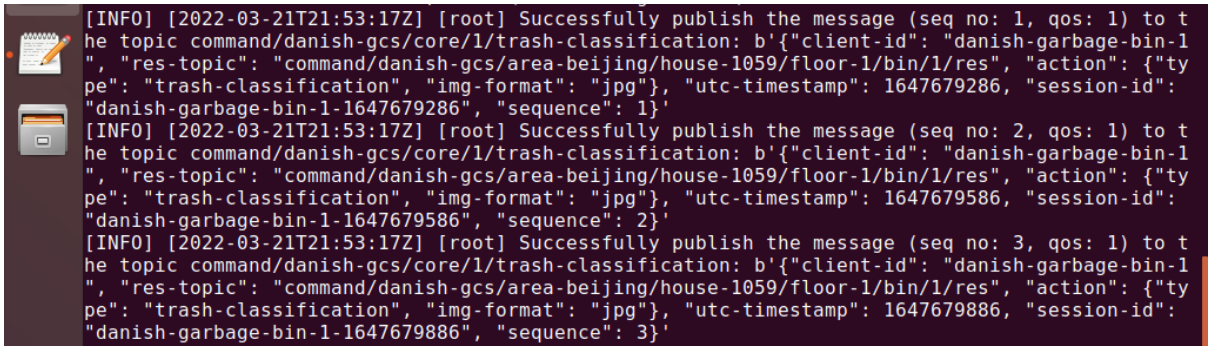
Argument name	Description
thing-name	The thing name that is used to authenticate with the Moquette MQTT broker.
dev-cert	The device's certificate that is used for authentication.
dev-key	The device's private key that is used for authentication.
root-ca	The root CA or CA certificate to verify the device's certificate. In this assignment, Amazon Root CA 1 is used as verification certificate for the Amazon-signed client devices' certificates.
pub-topic	The topic to publish the messages to.
sub-topic	The topic to receive the messages from.
mode	Enable publish on "pub" mode or "both" mode; enable subscribe on "sub" mode or "both" mode. Possible values are "pub", "sub", "both".
seq-no	The start sequence number of the messages to be published.
qos	Quality of service used in publishing/subscribing messages. Possible values are "at_most_once" and "at_least_once". "exactly_one"/QOS 2 is not supported by AWS IoT MQTT protocols.
sub-count	Number of messages expected to be received from the subscribed topic.
messages	A file containing files a list of inference requests generated by executing *python3 ./client_device/create_inference_req.py*.
verbosity	The verbosity level used in logging. Possible values are "nologs", "fatal", "error", "warn", "info", "debug", "trace".

4.9.1 Validate Command's Logs from Garbage Bin/Client Device

1) The full sample log file of the command can be found in *logs/pub_sub_to_core_device-Success.log*.

2) The logs indicate that the client device has successfully subscribe to the topic to receive the inference responses from the core device. Based on the logs, MQTT server granted QOS of 1. Note that the MQTT server might not grant the exact QOS value requested by the client device.

3) The logs indicate that the client device has successfully send the inference requests to the core device.



4) The logs indicate that the client device has successfully received the inference responses from the core device.

4.9.2 Validate Component's Logs from Core Device

1) The program has logged messages to the **danish.GCS.TrashClassification.log** located in the **/greengrass/v2/logs** directory**.** The full sample log file of the component can be found in *logs/danish.GCS.TrashClassification-Success.log.*

2) Run the command below to view the logs.

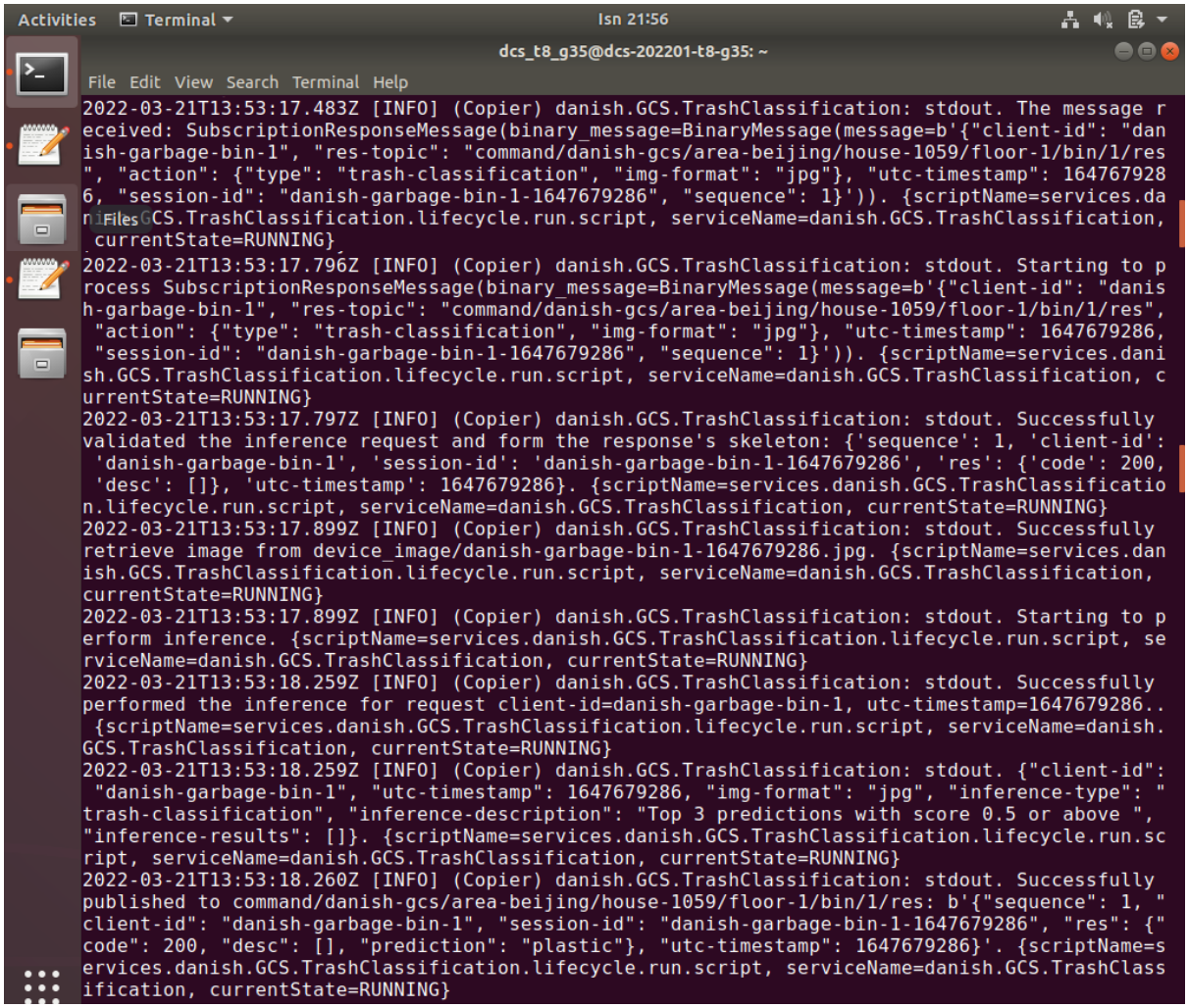
```
sudo cat /greengrass/v2/logs/danish.GCS.TrashClassification.log
```

3) For the remaining section, the important sections in the log file are discussed to ensure that the readers understand the healthy behaviour of the **TrashClassification** component.

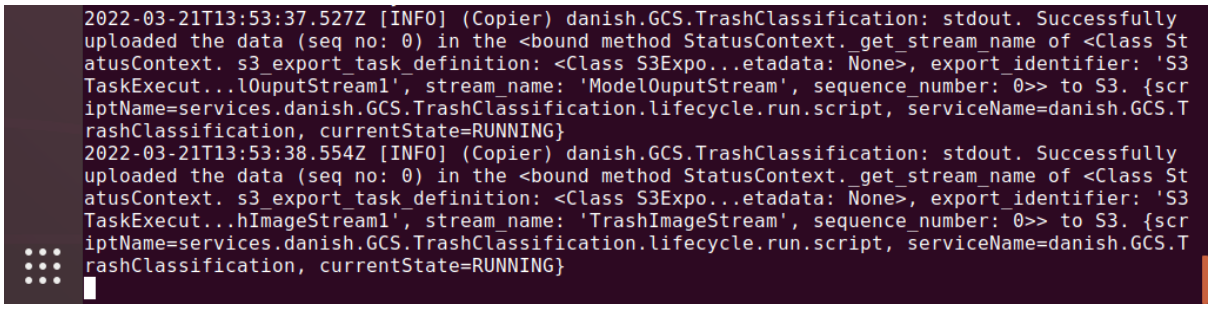
4) To ensure the component has successfully installed and initialized, ensure that the inference component has successfully:

- Created IPC client to subscribe and publish to and from MQTT topics.
- Subscribed to the component's configuration changes to fetch the latest configuration from the component recipe.
- Created the streams to export model outputs and images to the S3 bucket.
- Subscribed to the user-defined topic to receive inference requests from the client device.

5) The logs below indicate that the component has successfully received the correctly formatted inference request, performed the inference, and successfully published the inference response to the user-defined topic.

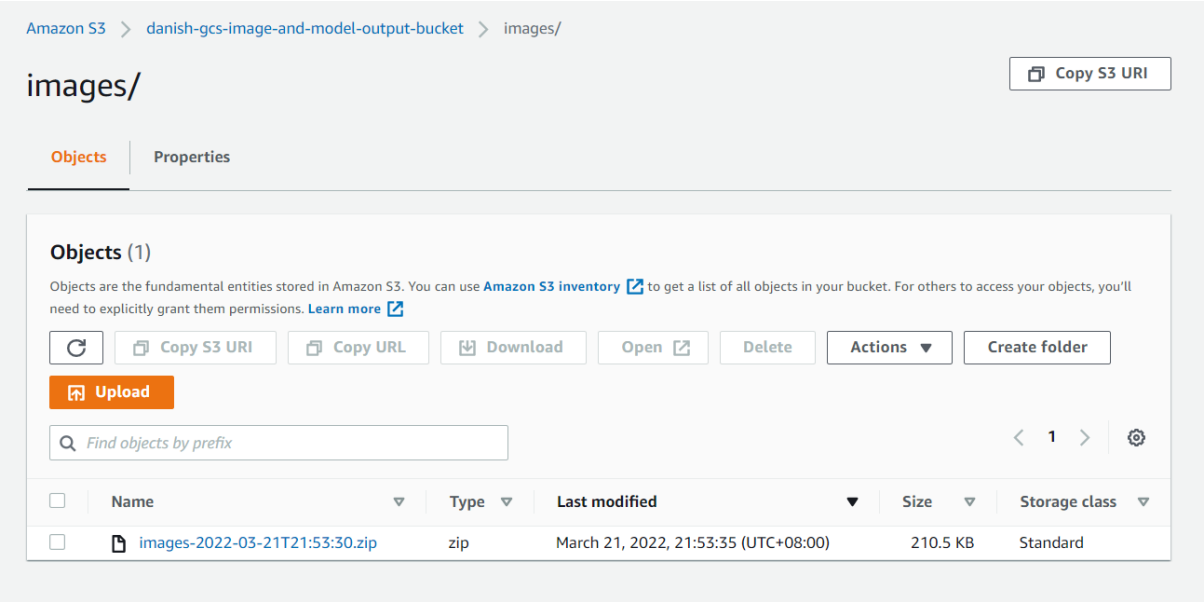


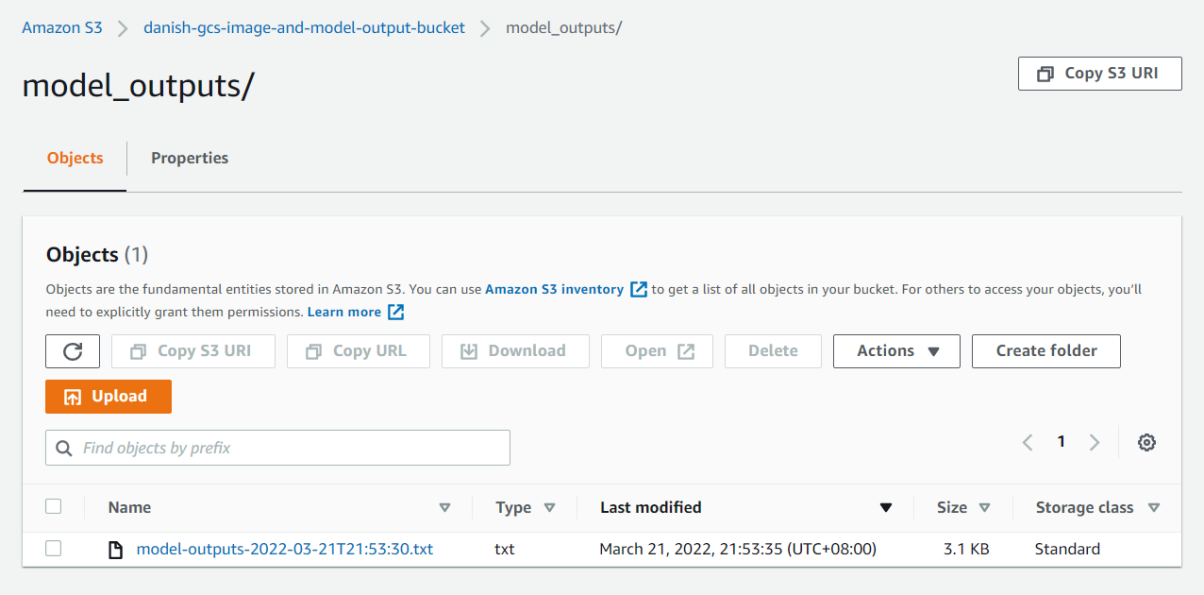
6) The logs below indicate that the inference component has successfully uploaded the model outputs to the **ModelOutputStream** and uploaded the images to the **TrashImageStream** to the S3 bucket, respectively. If the component failed to upload the data (which do rarely occur), the component will later export the old data with the new data after performing the next twelve inferences.



4.9.3 Validate Results from AWS Management Console

1) Check that the images and model outputs have successfully uploaded to the user defined S3 bucket. Check for errors in the component's log file located in the core device if the data is not seen in the S3 bucket.



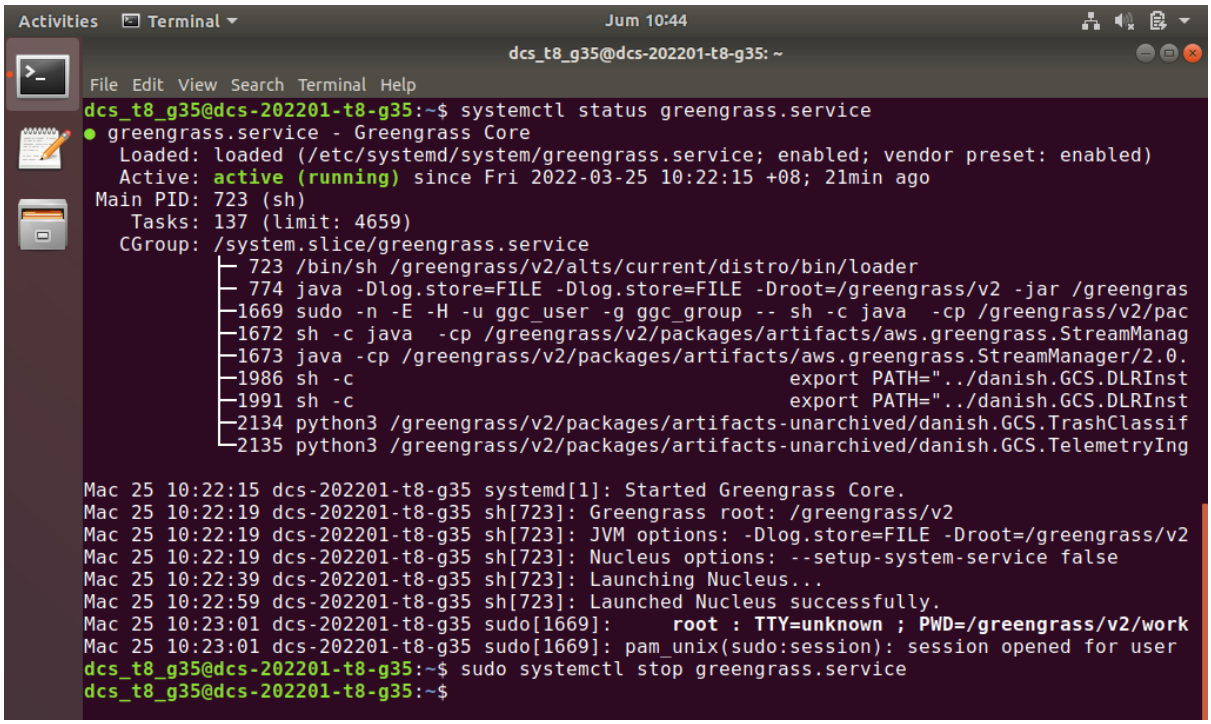


2) The example files can be found in *s3_output_files* folder.

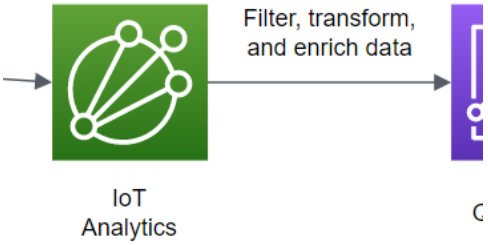
4.10 Shutdown Greengrass System Service

1) Before shutting down the core device's virtual machine, it is crucial to execute the command below to properly terminate the Greengrass system service. If not, some of the AWS public components like Stream Manager component will not shutdown properly and will cause the **danish.GCS.TrashClassification** and **danish.GCS.TelemetryIngestion** to be in the **Broken** state.

```
sudo systemctl stop greengrass.service
```

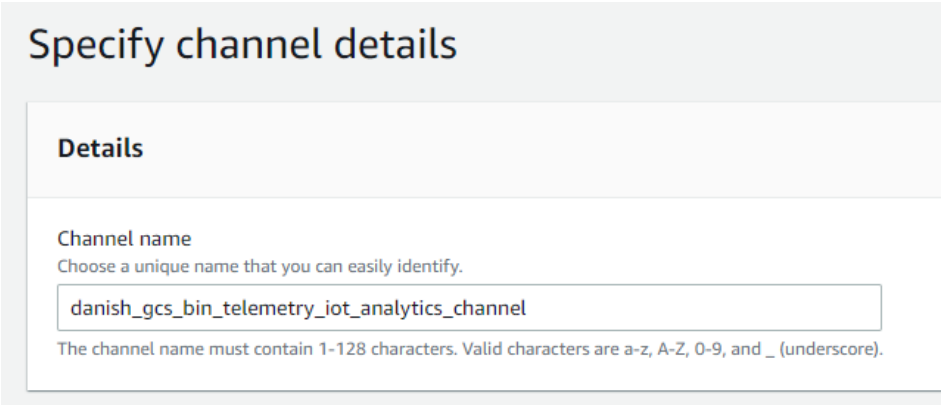


Section 5: AWS IoT Analytics

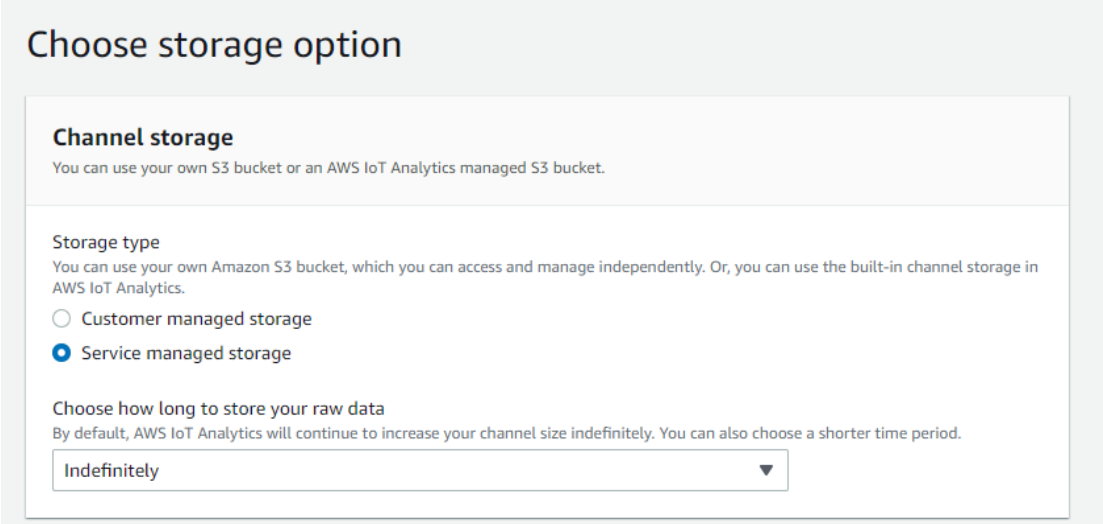


5.1 Create Channel

1) In the **AWS IoT Analytics Console** navigation menu, select **Channels** > **Create channel**. Name the channel as **danish_gcs_bin_telemetry_iot_analytics_channel**. Select **Next**.



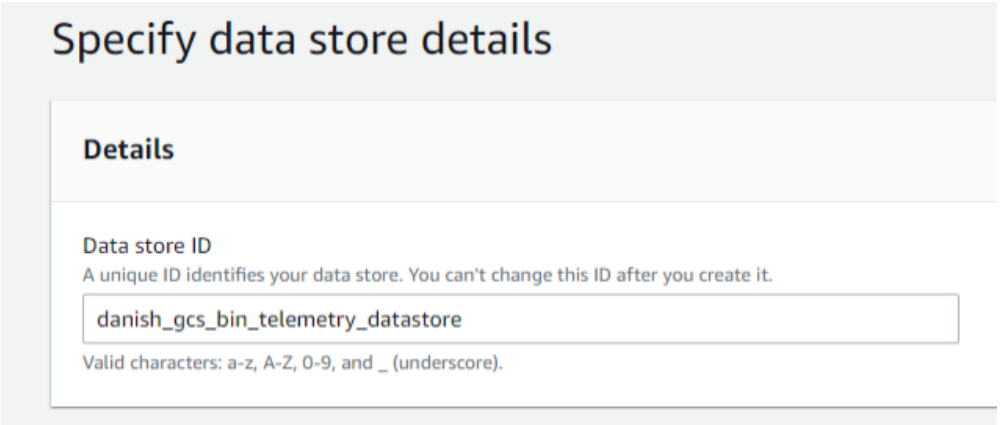
2) Select **Service managed storage**. Select **Indefinitely** in the drop-down menu. Select **Next**.



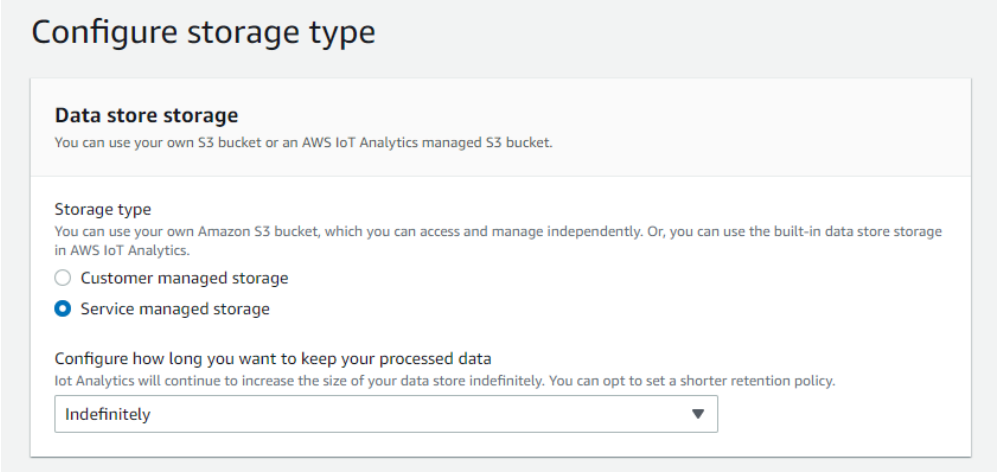
3) Skip **Step 3: Configure source**. Select **Next**. Review and select **Create channel**.

5.2 Create Data Store

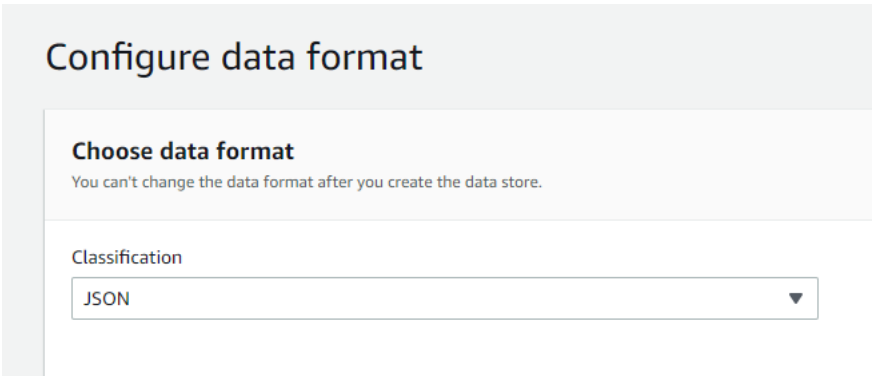
1) In the **AWS IoT Analytics Console** navigation menu, select **Data stores > Create data store**. Name the data store as **danish_gcs_bin_telemetry_datastore**. Select **Next**.



2) Select **Service managed storage**. Select **Indefinitely** in the drop-down menu. Select **Next**.



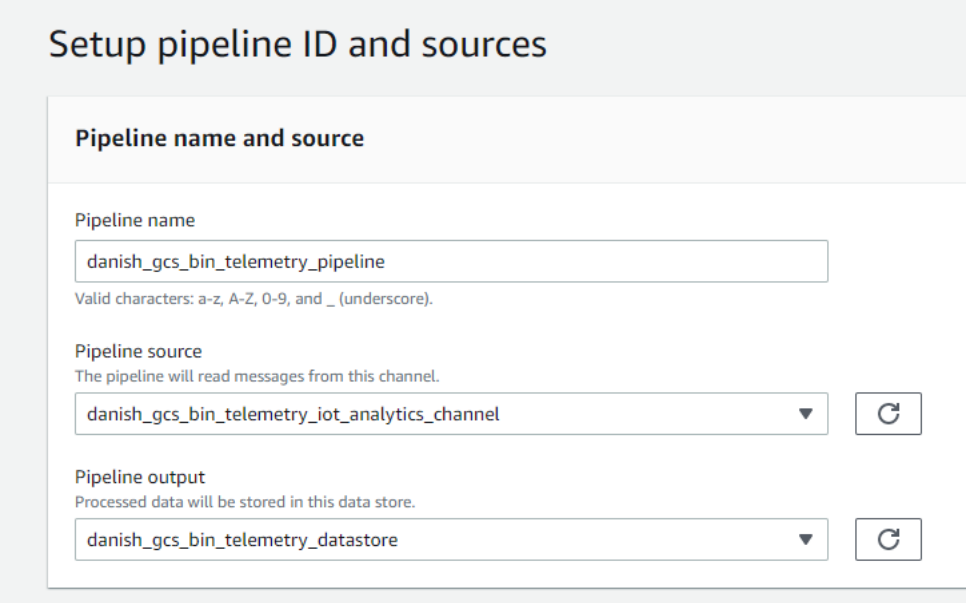
3) Select **JSON** in the drop-down menu. Select **Next**.



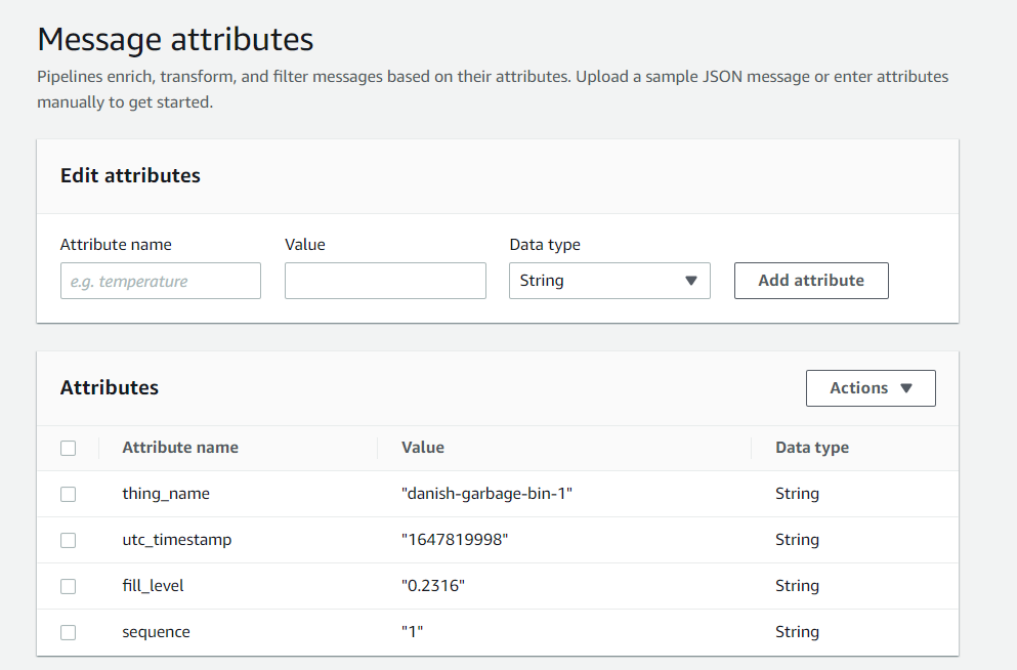
4) Skip **Step 4: Add data partitions**. Select **Next**. Review and select **Create data store**.

5.3 Create Pipeline

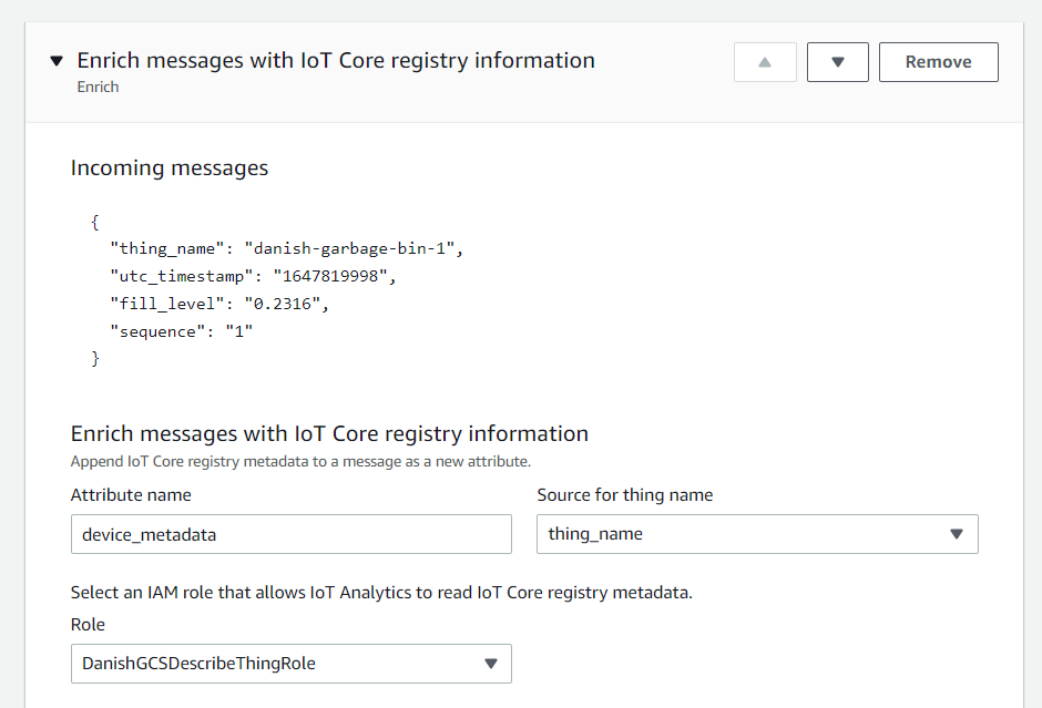
1) In the **AWS IoT Analytics Console** navigation menu, select **Pipelines > Create pipeline**. Name the pipeline as **danish_gcs_bin_telemetry_pipeline**. In the **Pipeline source** and the **Pipeline output**, select the newly created channel and data store, respectively. Select **Next**.



2) Manually add message attribute as shown in the image below. Select **Next**.

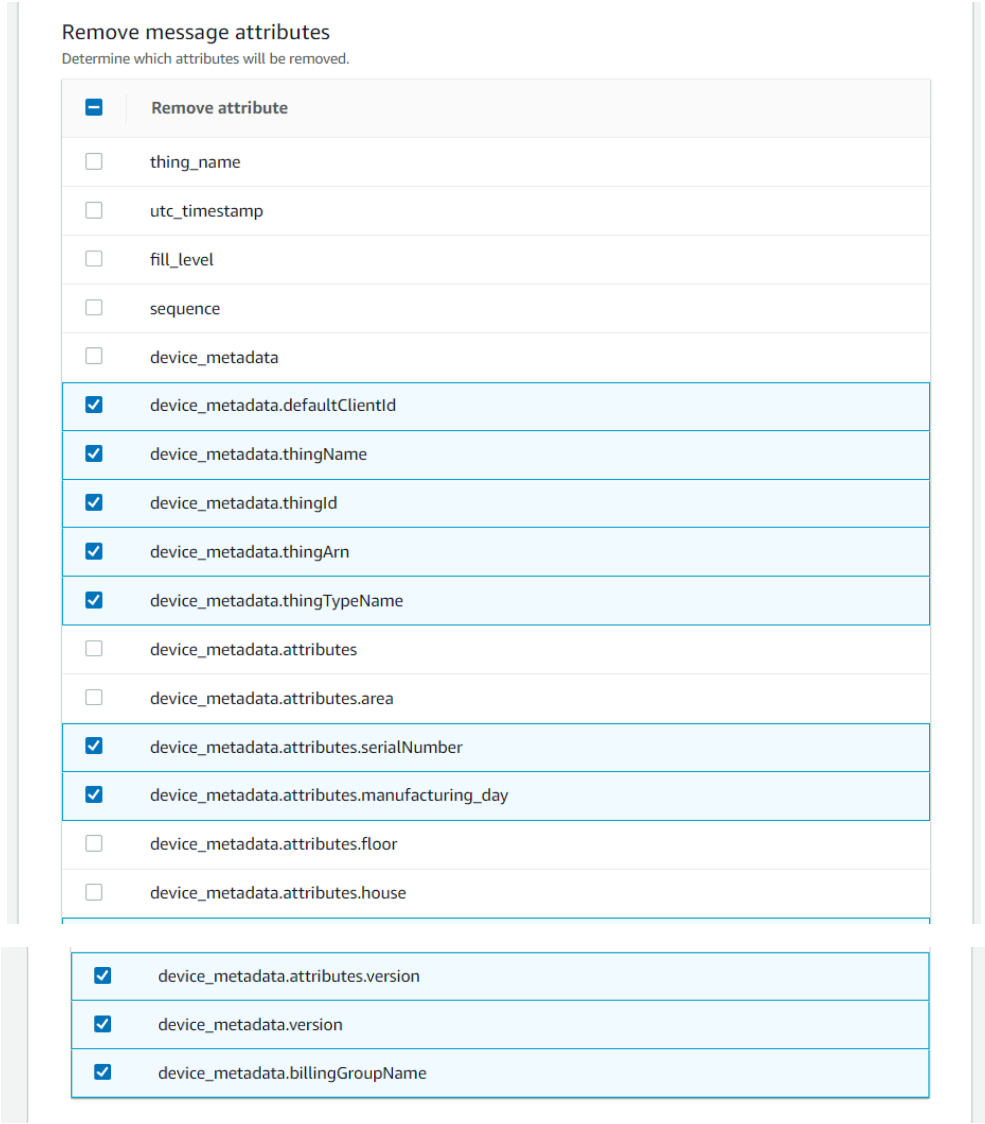


3) Add the first pipeline activity. Select **Enrich messages with IoT Core registry information** from the drop-down menu. Configure the activity as shown in the image below. Ensure that the IAM service role named **DanishGCSDescribeThingRole** is created.



4) The outgoing message of the first pipeline activity is shown as below.

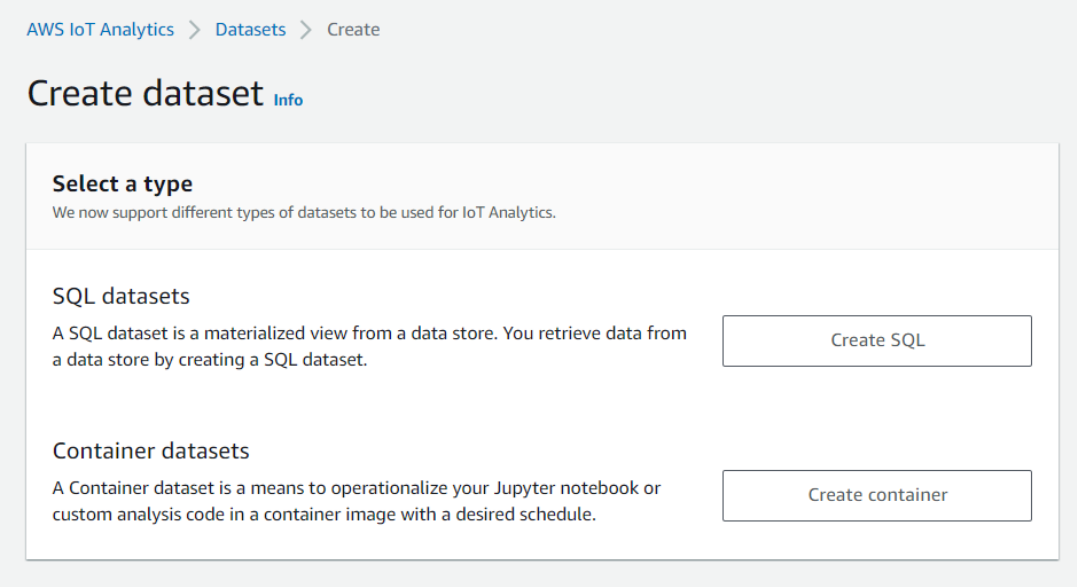
4) Add the second pipeline activity. Select **Remove message attributes**. Configure the activity as shown in the image below.



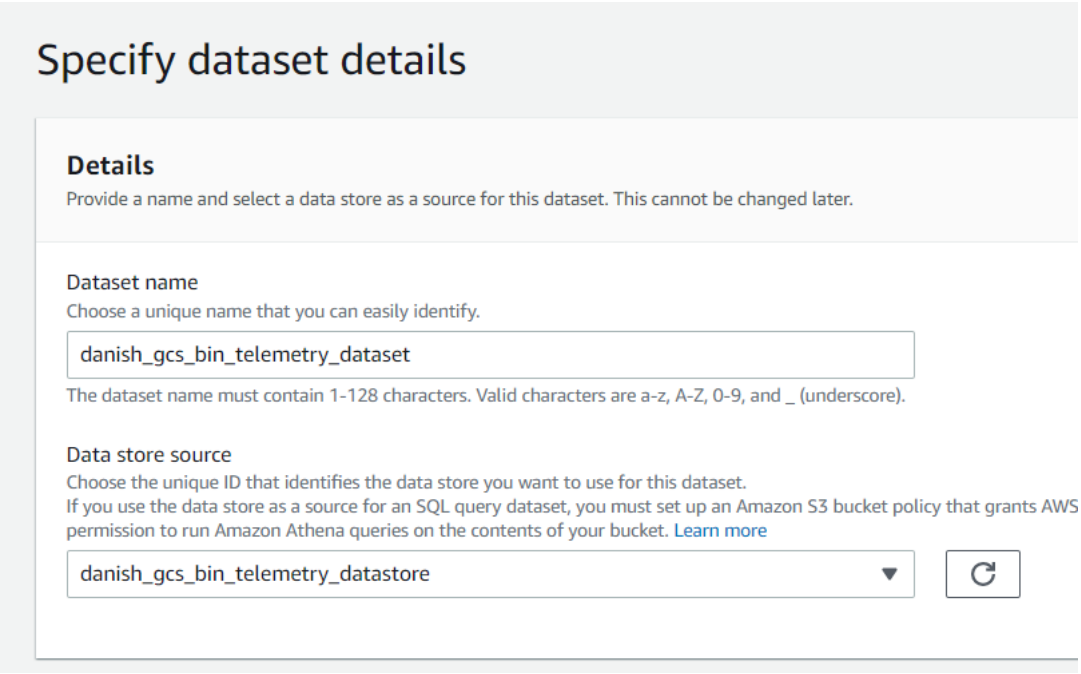
5) The enriched and transformed message should look like the one in the image below. Select **Next**. Review and select **Create pipeline**.

5.4 Create Dataset

1) In the **AWS IoT Analytics Console** navigation menu, select **Datasets** > **Create dataset**. Select **Create SQL**.

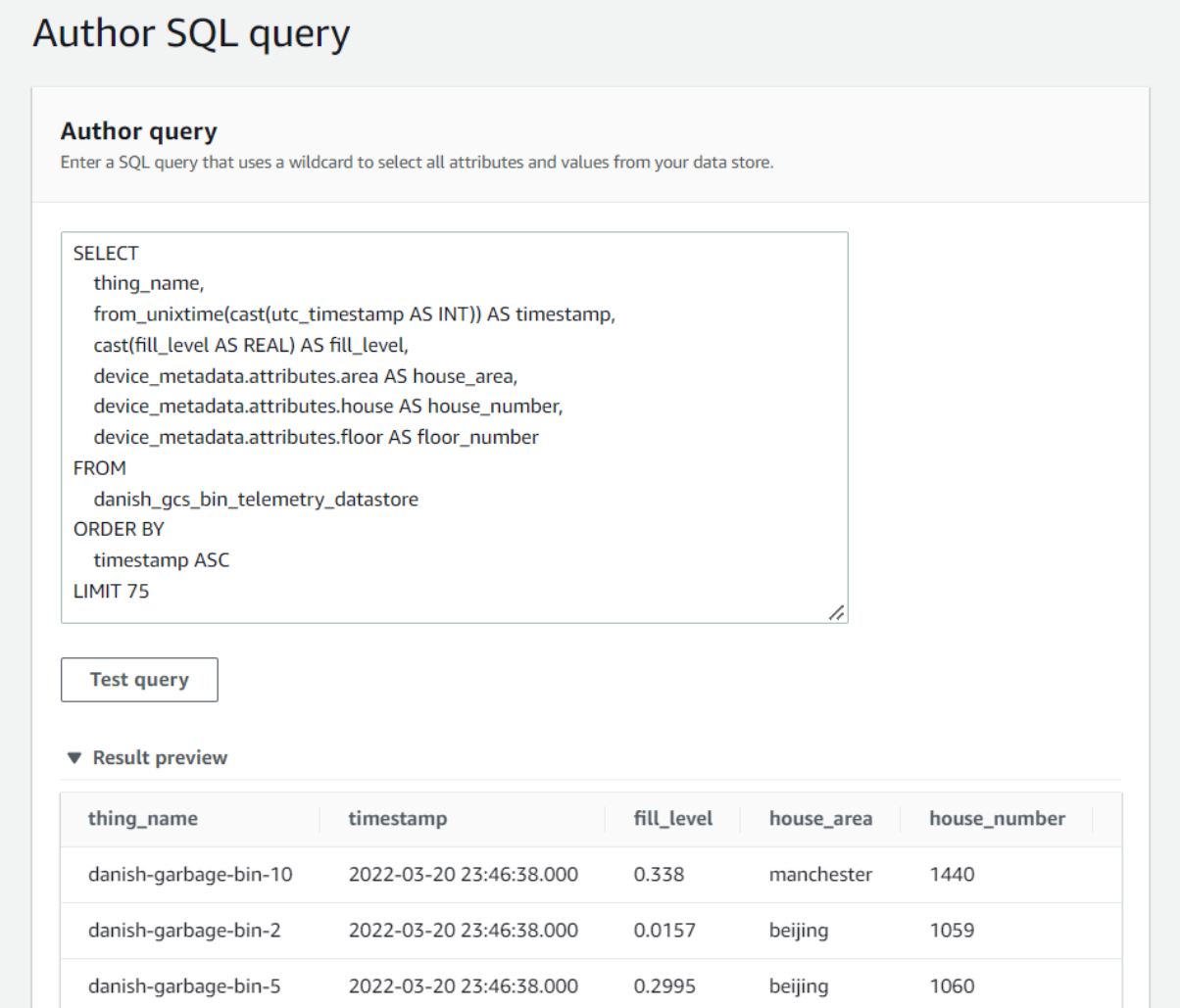


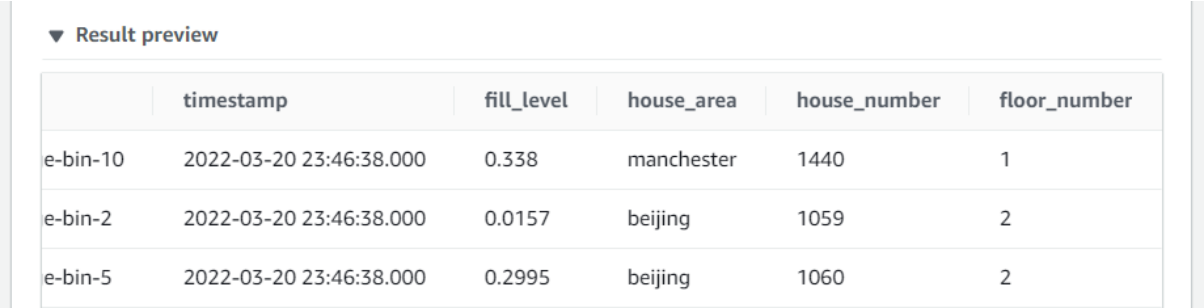
2) Name the dataset as **danish_gcs_bin_telemetry_dataset**. Select the newly created data store in the drop-down menu. Select **Next**.



3) In the **Step 2: Author SQL query,** enter the SQL query as shown below. Select **Test query** to check if the query is working.

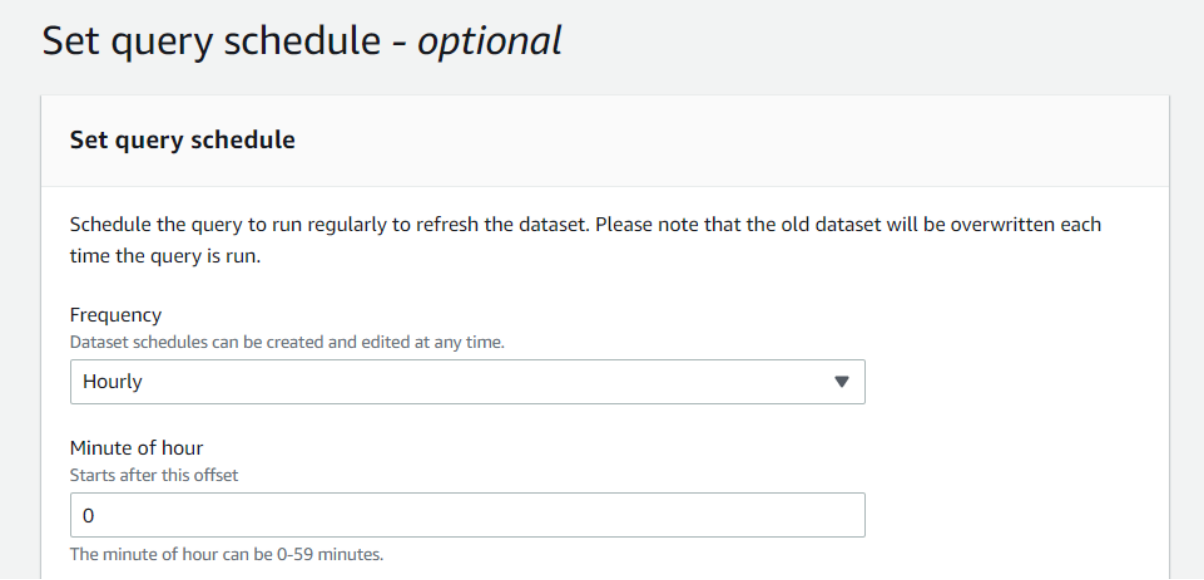
```
SELECT
   thing_name,
   from_unixtime(cast(utc_timestamp AS INT)) AS timestamp,
   cast(fill_level AS REAL) AS fill_level,
   device_metadata.attributes.area AS house_area,
   device_metadata.attributes.house AS house_number,
   device_metadata.attributes.floor AS floor_number
FROM
   danish_gcs_bin_telemetry_datastore
ORDER BY
   timestamp ASC
LIMIT 75
```



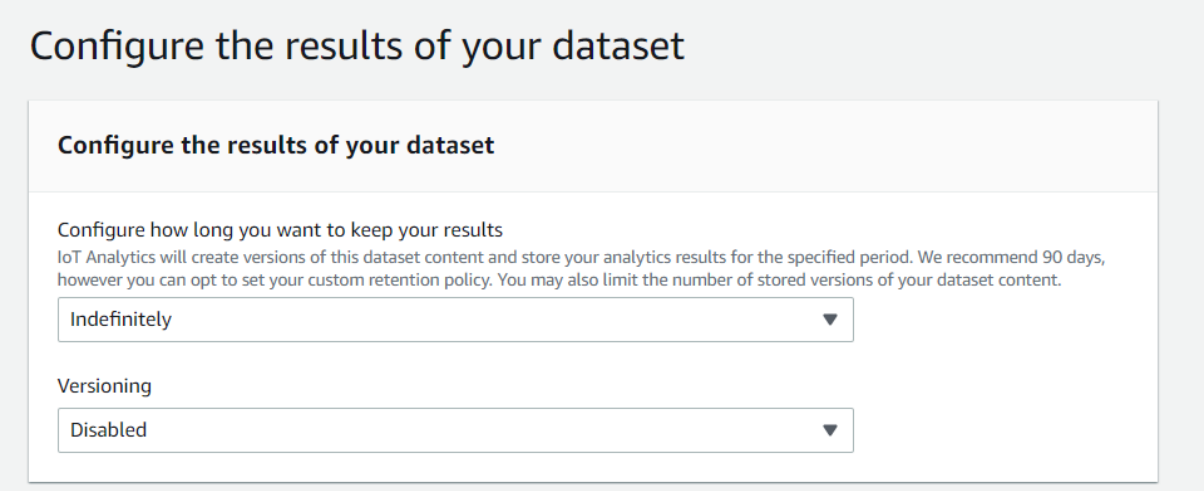


4) Skip **Step 3: Configure data selection filter**. Select **Next**.

5) In the **Step 4: Set query schedule**, set the frequency to **Hourly**. Select **Next**.

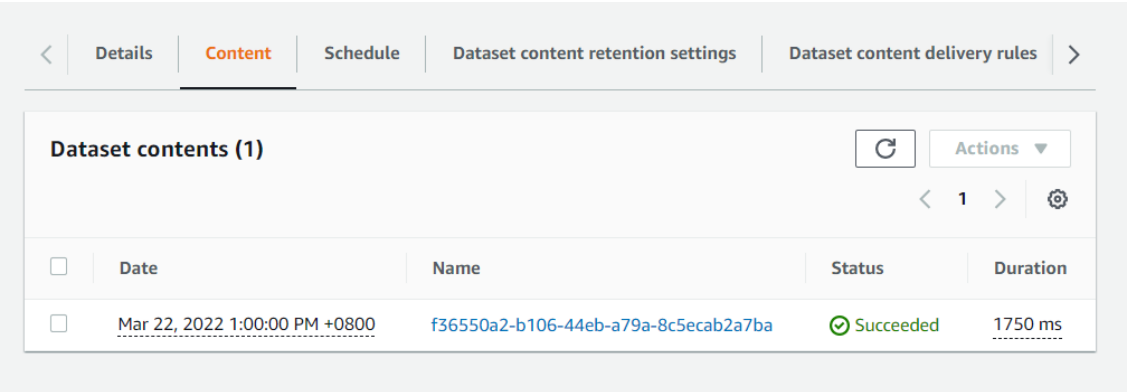


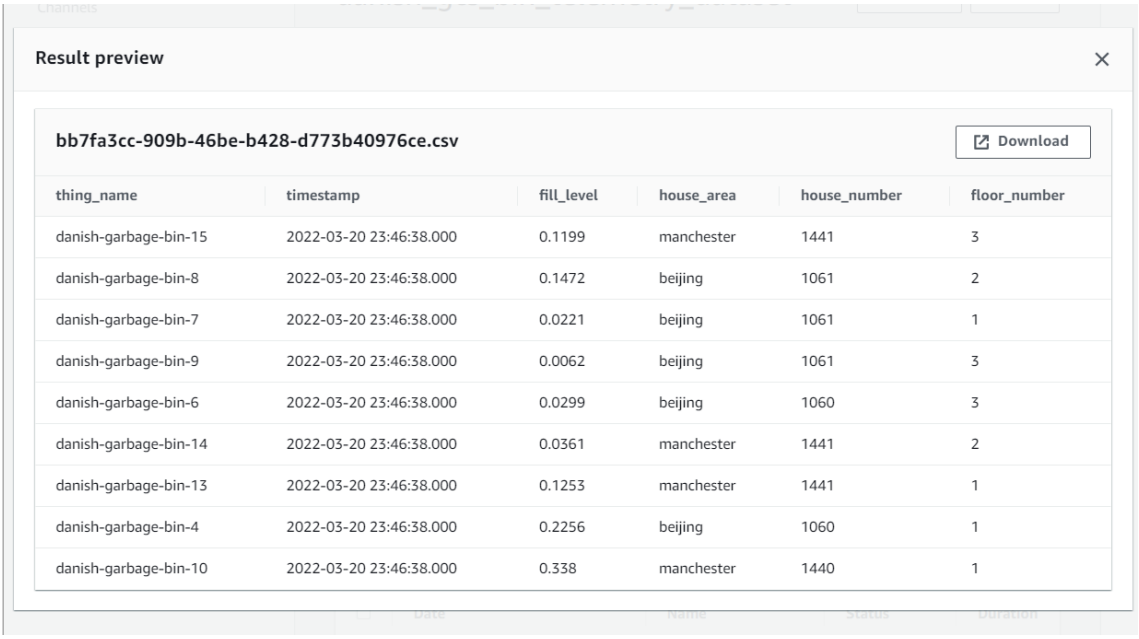
6) In the **Step 5: Configure the results of your dataset**, select the options as shown below.



7) Skip **Step 6: Configure dataset content delivery rules**. Select **Next**. Review and select **Create dataset**.

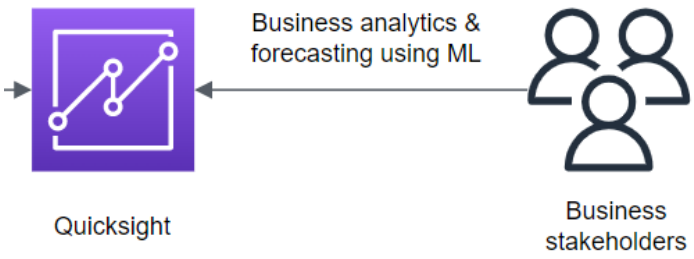
8) Select the **danish_gcs_bin_telemetry_dataset** in the table. As shown in the following images, Select **Run now** to retrieve data from the data store using SQL. Select the content name under **Content** section to view the result.



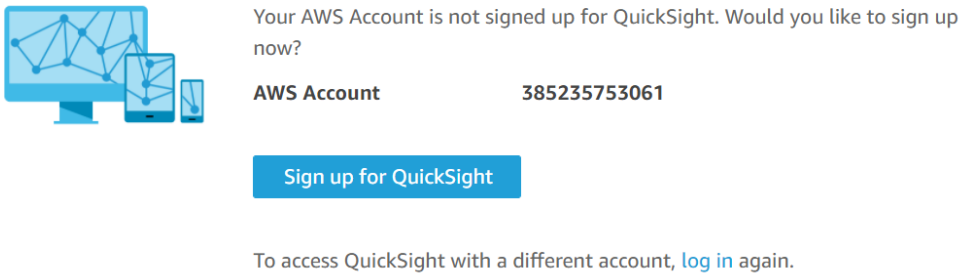


thing_name	timestamp	fill_level	house_area	house_number	floor_number
danish-garbage-bin-15	2022-03-20 23:46:38.000	0.1199	manchester	1441	3
danish-garbage-bin-8	2022-03-20 23:46:38.000	0.1472	beijing	1061	2
danish-garbage-bin-7	2022-03-20 23:46:38.000	0.0221	beijing	1061	1
danish-garbage-bin-9	2022-03-20 23:46:38.000	0.0062	beijing	1061	3
danish-garbage-bin-6	2022-03-20 23:46:38.000	0.0299	beijing	1060	3
danish-garbage-bin-14	2022-03-20 23:46:38.000	0.0361	manchester	1441	2
danish-garbage-bin-13	2022-03-20 23:46:38.000	0.1253	manchester	1441	1
danish-garbage-bin-4	2022-03-20 23:46:38.000	0.2256	beijing	1060	1
danish-garbage-bin-10	2022-03-20 23:46:38.000	0.338	manchester	1440	1

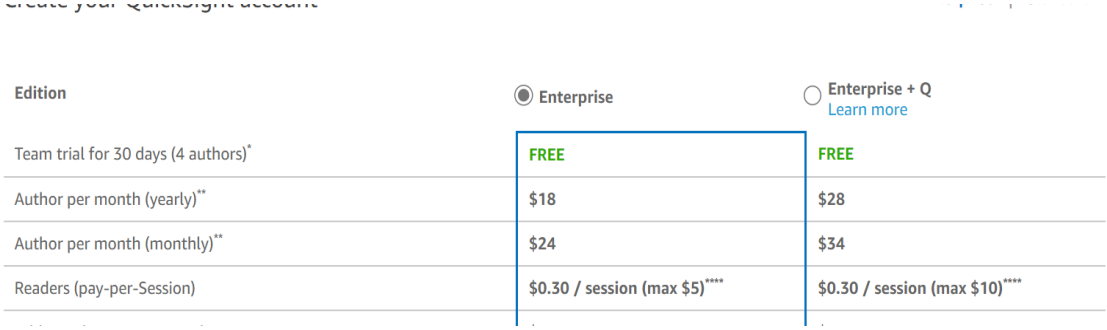
Section 6: Amazon QuickSight



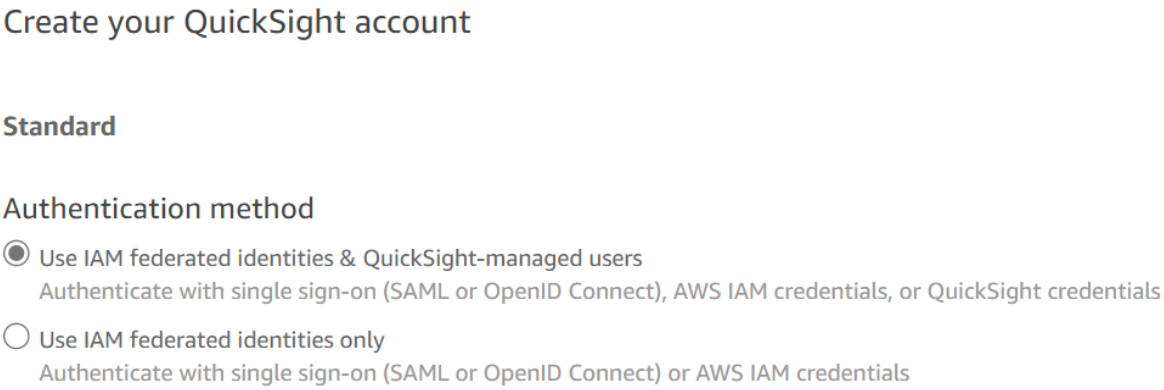
1) Navigate to QuickSight in AWS Management Console. Select **Sign up for QuickSight** to create a new account.

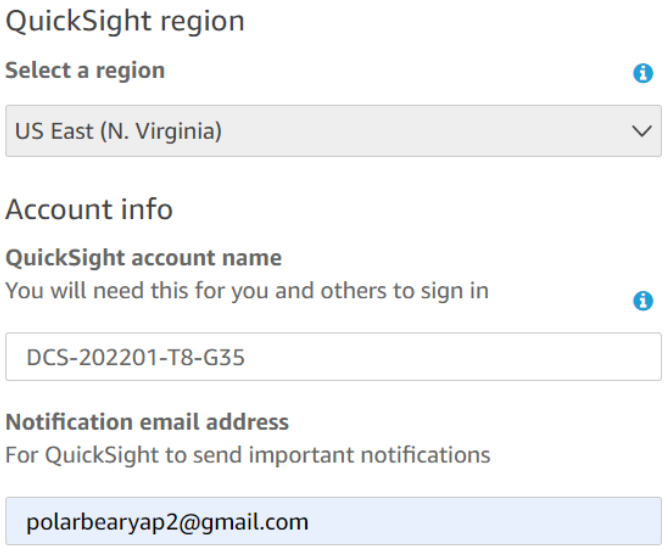


2) Select **Standard** at the top right corner. Enter details as shown in the following images.

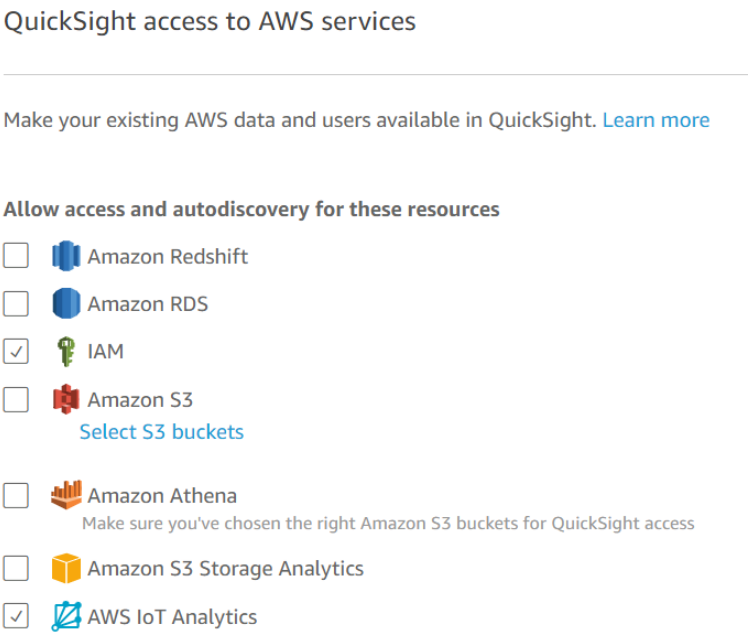


Edition	Enterprise	Enterprise + Q Learn more
Team trial for 30 days (4 authors)*	FREE	FREE
Author per month (yearly)**	$18	$28
Author per month (monthly)**	$24	$34
Readers (pay-per-Session)	$0.30 / session (max $5)****	$0.30 / session (max $10)****

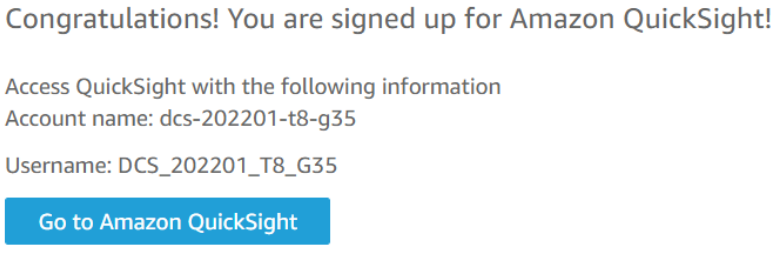




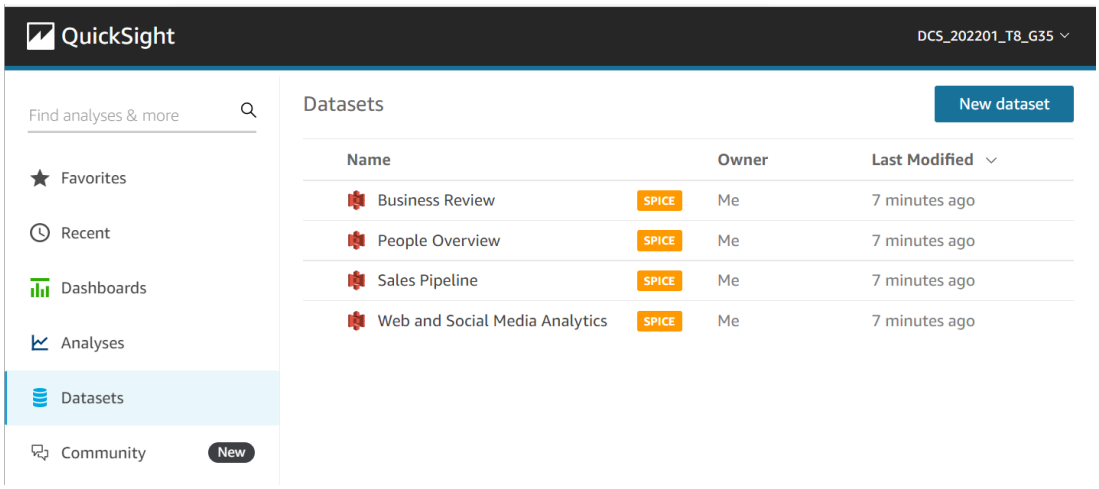
3) For this assignment, select **IAM** and **AWS IoT Analytics**.

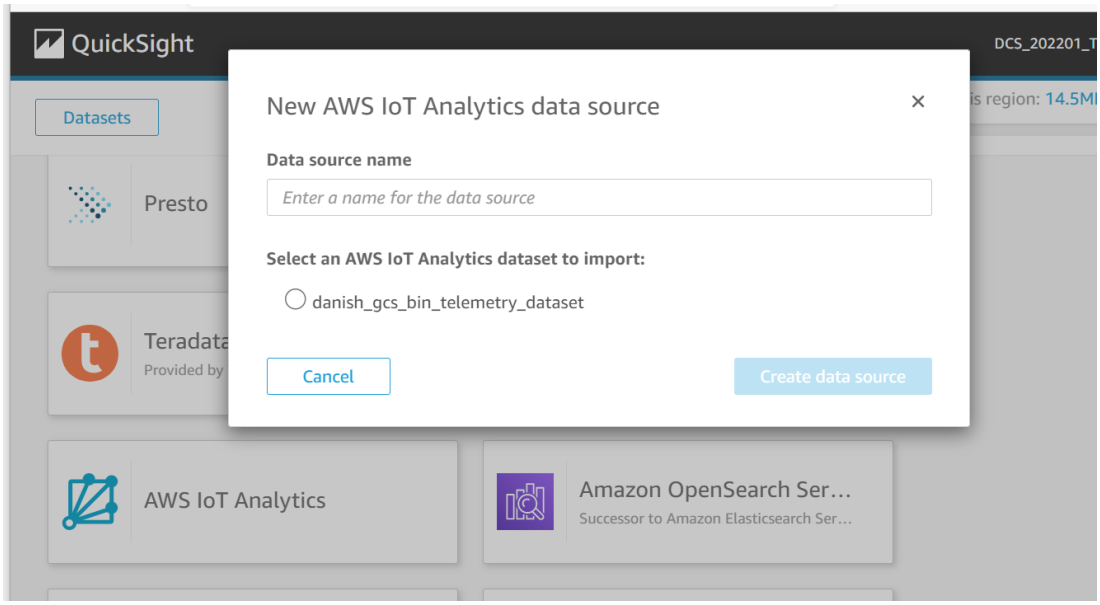


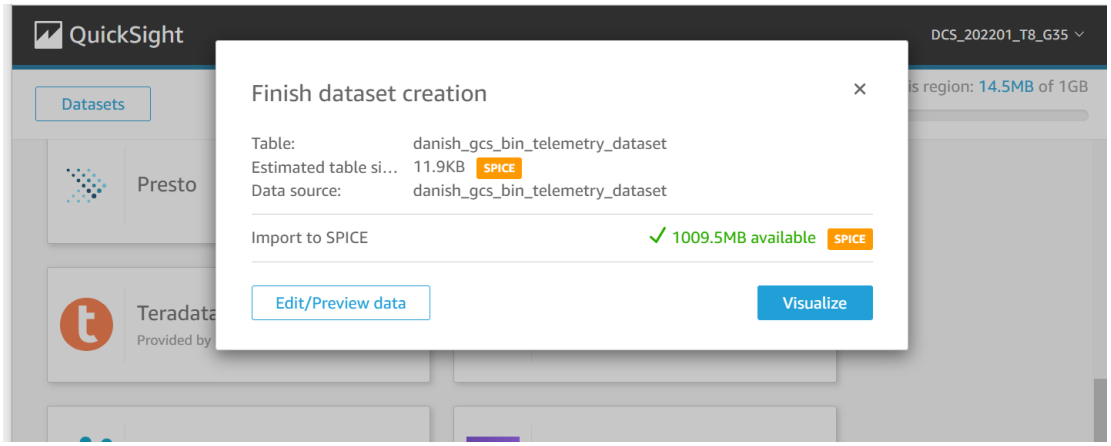
4) Select **Go to Amazon QuickSight**.



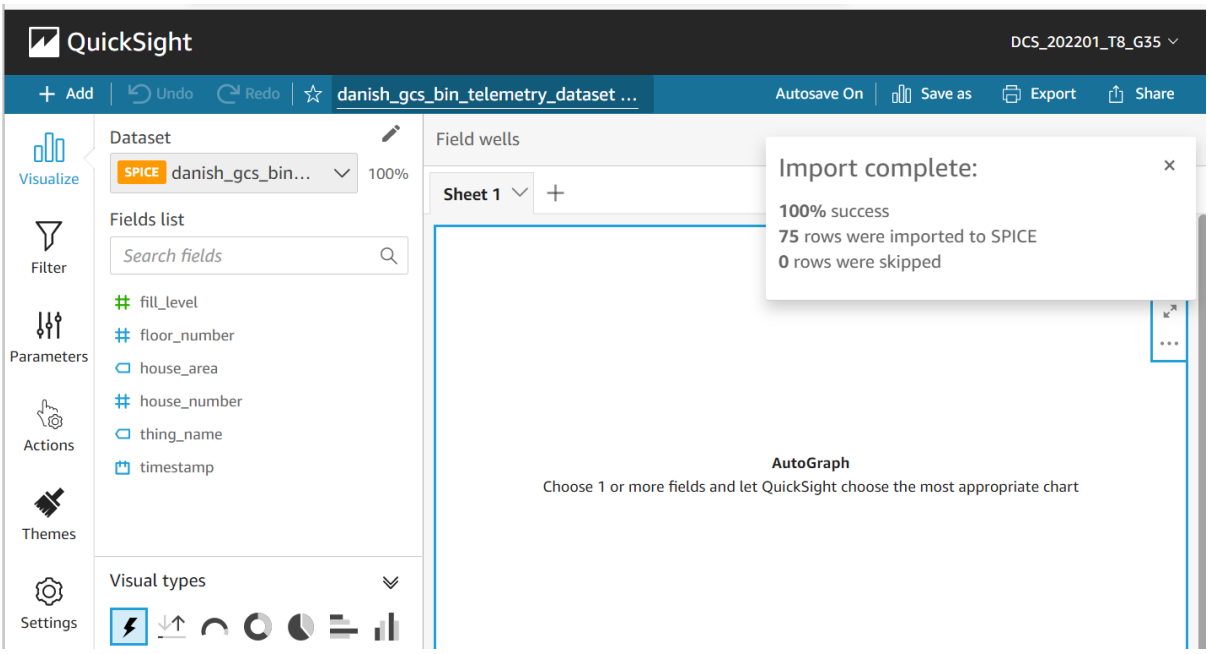
5) Select **Datasets** > **New dataset** > **AWS IoT Analytics**. Select **danish_gcs_bin_telemetry_dataset** as the new AWS IoT Analytics data source. Then, select **Visualize** as shown in the image below.







6) With the imported fill level telemetry, the business stakeholder can now perform business analytics and forecasting on QuickSight.



Section 7: Developer Guide

7.1 Local Deployment of Greengrass Component

1) To speed up development and testing of custom Greengrass component, execute the following commands every time a new terminal is opened. Replace the **COMPONENT_NAME** with your current component name. Update the **RECIPE_DIR** and **ARTIFACT_DIR**. Keep the **COMPONENT_VERSION** at 1.0.0. This is because `gg_local_delete` will completely remove the component so that the component can re-deploy with the same version number.

```
export COMPONENT_NAME="danish.GCS.TelemetryIngestion.DevTest"
export COMPONENT_VERSION="1.0.0"
export RECIPE_DIR="/home/dcs_t8_g35/greengrass-
dev/components/${COMPONENT_NAME}/recipe"
export ARTIFACT_DIR="/home/dcs_t8_g35/greengrass-
dev/components/${COMPONENT_NAME}/artifacts"

#Deploy-Helper-Function
gg_local_deploy(){
sudo /greengrass/v2/bin/greengrass-cli deployment create \
--recipeDir $RECIPE_DIR --artifactDir $ARTIFACT_DIR \
--merge "$COMPONENT_NAME=$COMPONENT_VERSION";
}

#Remove-Helper-Function
gg_local_delete(){
sudo /greengrass/v2/bin/greengrass-cli --ggcRootPath /greengrass/v2
deployment create \
--remove "$COMPONENT_NAME";
}
```

2) Execute **gg_local_deploy** to deploy the component locally and execute **gg_local_delete** to delete the component locally. Both functions will return the **deployment-id** and can be used to track individual deployment status. To check the deployment status of your local component, open a separate terminal to execute the first two commands to view the live deployment logs, respectively. Execute the last two commands to check the final deployment status.

```
sudo tail -f /greengrass/log/greengrass/greengrass.log
sudo tail -f /greengrass/log/greengrass/<your-local-component-name>.log
sudo /greengrass/v2/bin/greengrass-cli deployment status -i <deployment-id>
sudo /greengrass/v2/bin/greengrass-cli component list | grep -E
'^Component'
```

3) Below are some useful development tips from the author:

Development Tips #1: Omit the **Artifacts** in the component recipe. If you specify this, then the greengrass-cli will fetch the artifacts from the S3 bucket instead from the current device's local directory.

```
},
"Artifacts": [
    {
        "Uri": "s3://danish-gcs-model-artifacts-bucket/v1/aws.greengrass.
        DLRImageClassification.artifacts/trash-cf-inference.zip",
        "Digest": "D5HQB8L2haURI1NiOAr5E2UtZtqWZy1EkWGb78Oszzc=",
        "Algorithm": "SHA-256",
        "Unarchive": "ZIP",
        "Permission": {
            "Read": "OWNER",
            "Execute": "NONE"
        }
    }
```

Development Tips #2: Change the file path of the artifacts accordingly. For example, the main.py is at **{artifacts:path}/main.py** during the local deployment but the main.py is at **{artifacts: decompressedPath}/main.py** for cloud deployment.

```
"run": {
    "RequiresPrivilege": "true",
    "script": "\n                                    export PATH=\"
    {danish.GCS.DLRInstallation:configuration:/MLRootPath}/
    greengrass_ml_dlr_conda/
    bin:$PATH\"\n                                    eval \"$({danish.
    GCS.DLRInstallation:configuration:/MLRootPath}/
    greengrass_ml_dlr_conda/bin/conda shell.bash hook)
    \"\n                                conda activate
    greengrass_ml_dlr_conda\n                                    python3
    {artifacts:path}/main.py"
}
```

Development Tips #3: The directory structure below shows one way to organize the recipes and artifacts for multiple local components.

/home/**your-username**/greengrass-dev/components
- > **your-local-component-name**
  - > recipe
    - > *your-component-recipe-here*
  - > artifacts
    - > **your-local-component-name** (child directory must be a version number)
      - > 1.0.0
        - > *all-of-your-artifacts-here*
- > **your-second-local-component-name**
  - > recipe
    - > *your-second-component-recipe-here*
  - > artifacts
    - > **your-second-local-component-name**
      - > 1.0.0
        - > *all-of-your-second-artifacts-here*

7.2 Communication between Client Devices with Core Device and AWS IoT Core

To enable communication **(1)** between client devices and Greengrass custom components and **(2)** between client devices and AWS IoT Core (Amazon Web Services, Inc., 2022e), the following Greengrass public components must be deployed to the core device:

a) Moquette MQTT broker component (aws.greengrass.clientdevices.mqtt.Moquette): The local Moquette MQTT broker that handles messages between client devices and a core device.
b) MQTT bridge component (aws.greengrass.clientdevices.mqtt.Bridge): Relay and route MQTT messages between the three message brokers which are:
    i. Local Moquette MQTT broker (**LocalMqtt**): Handle messages between client devices and a core device.
    ii. Local publish/subscribe messaging IPC service (**Pubsub**): Handle messages between components within a core device.
    iii. AWS IoT Core MQTT broker (**IotCore**): Handle messages between IoT devices and AWS cloud destinations.
c) Client device auth component (aws.greengrass.clientdevices.Auth): Authenticate and authorize client devices to connect to the local Moquette MQTT broker, and publish or subscribe to MQTT topics.

7.3 Inference Component's Implementation Details

7.3.1 Component Dependencies

Inference component is highly dependent on some of the components as shown below. As a result, the **DependencyType** is set to **HARD** in the component recipe so that the inference component restarts if any of the dependencies changes state.

a) danish.GCS.TrashClassification.ModelStore: This component is required to provide the trash classification model needed to perform trash classification inference.

b) danish.GCS.DLRInstallation: This component is required to manage the Conda environment and install the Python dependencies needed to run the component. Note that this component is shared with data ingestion component.

c) MQTT bridge component: This component is required to enable communication between client devices and the core device through bridging MQTT topic mappings.

d) Stream manager component: This component is required to set up a stream manager client connection to export batches of data to the S3 bucket.

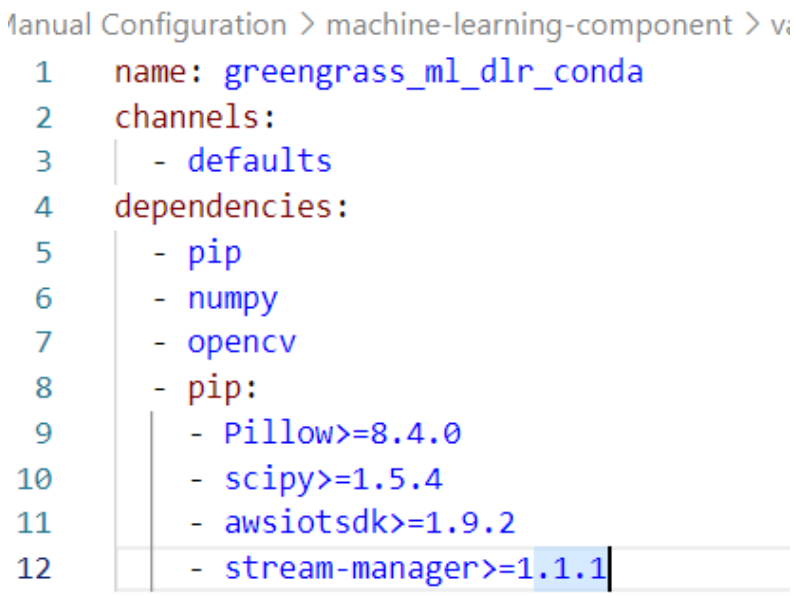
e) IP Detector component and Greengrass nucleus component: These components are required to start before the inference component can start.

```
},
"ComponentDependencies": {
    "danish.GCS.TrashClassification.ModelStore": {
        "VersionRequirement": ">=1.0.0 <2.0.0",
        "DependencyType": "HARD"
    },
    "danish.GCS.DLRInstallation": {
        "VersionRequirement": ">=1.0.0 <2.0.0",
        "DependencyType": "HARD"
    },
    "aws.greengrass.StreamManager": {
        "VersionRequirement": ">=2.0.14",
        "DependencyType": "HARD"
    },
    "aws.greengrass.clientdevices.IPDetector": {
        "VersionRequirement": ">=2.1.1",
        "DependencyType": "SOFT"
    },
    "aws.greengrass.clientdevices.mqtt.Bridge": {
        "VersionRequirement": ">=2.1.0",
        "DependencyType": "HARD"
    },
    "aws.greengrass.Nucleus": {
        "VersionRequirement": ">=2.0.0 <2.6.0",
        "DependencyType": "SOFT"
    }
},
```

7.3.2 Component's Artifact's Dependencies

Below are the Python packages that are required in the inference component:

a) pip: To install Python packages using pip.
b) numpy: Required to perform inference.
c) opencv: Required to preprocess images.
d) Pillow: Required to load images.
e) scipy: Required to apply SoftMax functions on the trash classifiers' output to get the class probabilities.
f) awsiotsdk: Required to set up an IPC client connection to perform IPC operations such as subscribe to topics, receive messages from subscribed topics, publish messages to topics, and fetch component configuration's updates.
g) stream-manager: Required to set up a stream manager client connection to perform operations such as creating streams, appending message to the stream, and checking stream's upload status.



The file content above can be found in *s3_artifacts/variant.DLR.artifacts/environment.yaml.*

7.3.3 Sending Inference Request

PUBLISH: command/danish-gcs/core/1/trash-classification

COMMAND REQUEST PAYLOAD:

```
{
   "sequence": 1,
   "client-id": "danish-garbage-bin-1",
   "utc-timestamp": 1647679286,
   "session-id": "danish-garbage-bin-1-1647679286",
   "res-topic": "command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res",
   "action": {
      "type": "trash-classification",
      "img-format": "jpg",
   }
}
```

1) To perform inference, the client device connects to the local Moquette MQTT broker and publishes an inference request to the topic named **command/danish-gcs/core/1/trash-classification**. Since the inference component subscribes to this topic via local publish/subscribe messaging IPC service, the component will receive the inference request. To enable such communication to happen, the following component recipes must be configured as shown below:

a) MQTT bridge component: A mapping is set up such that the MQTT bridge component listens for messages on the **command/danish-gcs/core/+/trash-classification** topic filter from the local Moquette MQTT broker and publishes the same messages to the same topic in the Local publish/subscribe messaging IPC service.

```
},
"BinToInferenceComponentMapping": {
    "topic": "command/danish-gcs/core/+/trash-classification",
    "source": "LocalMqtt",
    "target": "Pubsub"
},
```

b) Client device auth component: A policy must be set up such that the client device, in this case, **danish-garbage-bin-1**, can connect to the local Moquette MQTT broker and publish the inference request to the topic named **command/danish-gcs/core/1/trash-classification**.

```
"danish-garbage-bin-policy": {
    "AllowConnect": {
        "statementDescription": "Allow client devices to connect to the Greengrass core device.",
        "operations": [
            "mqtt:connect"
        ],
        "resources": [
            "*"
        ]
    },
    "AllowPublish": {
        "statementDescription": "Allow client devices to publish images to a topic to request trash type
        predictions from the inference component.",
        "operations": [
            "mqtt:publish"
        ],
        "resources": [
            "mqtt:topic:command/danish-gcs/core/1/trash-classification"
        ]
    }
},
```

c) Inference component: In the component recipe, authorization policies must be defined to allow the inference component to perform the IPC operations. In this case, a policy named **danish.GCS.TrashClassification:pubsub:2** is defined to authorize the inference component to subscribe to a topic named **command/danish-gcs/core/1/trash-classification** to receive inference requests.

```
    "danish.GCS.TrashClassification:pubsub:2": {
        "policyDescription": "Allow the component to receive inference requests from the client devices.",
        "operations": [
            "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
            "command/danish-gcs/core/1/trash-classification"
        ]
    }
```

7.3.4 Receiving Inference Response

PUBLISH: command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res

COMMAND RESPONSE PAYLOAD:


```
{
   "sequence": 1,
   "client-id": "danish-garbage-bin-1",
   "utc-timestamp": 1647679286,
   "session-id": "danish-garbage-bin-1-1647679286",
   "res": {
      "code": 200,
      "desc": [],
      "prediction": "plastic"
   }
}
```


1) Upon receiving the inference request, the core device will process the request and perform the inference. To ensure bidirectional communication, the inference request payload contains a field named **res-topic** that informs the core device which topic to publish the inference response to. In this case, the core device publishes an inference response to the topic named **command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res**. Since the client device is subscribed to this topic via local Moquette MQTT broker, the client device will receive the inference response. To enable such communication to happen, the following component recpies must be configured as shown below:

a) MQTT bridge component: A mapping is set up such that the MQTT bridge component listens for messages on the **command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res** topic from the Local publish/subscribe messaging IPC service and publishes the same messages to the same topic in the local Moquette MQTT broker. Note that if the **source** is **PubSub**, then MQTT topic wildcards cannot be used, so the topic names must be specified one by one as shown below.

```
"InferenceComponentToBin1Mapping": {
    "topic": "command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res",
    "source": "Pubsub",
    "target": "LocalMqtt"
},
"InferenceComponentToBin2Mapping": {
    "topic": "command/danish-gcs/area-beijing/house-1059/floor-2/bin/2/res",
    "source": "Pubsub",
    "target": "LocalMqtt"
},
"InferenceComponentToBin3Mapping": {
    "topic": "command/danish-gcs/area-beijing/house-1059/floor-3/bin/3/res",
    "source": "Pubsub",
    "target": "LocalMqtt"
},
```

b) Client device auth component: A policy must be set up such that the client device, in this case, **danish-garbage-bin-1**, can connect to the local Moquette MQTT broker and subscribe to the topic named **command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res** to receive the inference response.

```
"danish-garbage-bin-1-policy": {
    "AllowPublish": {
        "statementDescription": "Allow danish-garbage-bin-1 to publish fill level telemetry to the topic to
        be sent to various AWS cloud destinations.",
        "operations": [
            "mqtt:publish"
        ],
        "resources": [
            "mqtt:topic:data/danish-gcs/area-beijing/house-1059/floor-1/bin/1"
        ]
    },
    "AllowSubscribe": {
        "statementDescription": "Allow danish-garbage-bin-1 to subscribe to its own topic to receive
        predicted trash type.",
        "operations": [
            "mqtt:subscribe"
        ],
        "resources": [
            "mqtt:topicfilter:command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res"
        ]
    }
},
```

c) Inference component: In the component recipe, authorization policies must be defined to allow the inference component to perform the IPC operations. In this case, a policy named **danish.GCS.TrashClassification:pubsub:1** is defined to authorize the inference component to publish inference responses to a list of topics.

```
"accessControl": {
    "aws.greengrass.ipc.pubsub": {
        "danish.GCS.TrashClassification:pubsub:1": {
            "policyDescription": "Allow the component to send model predictions back to the client devices.",
            "operations": [
                "aws.greengrass#PublishToTopic"
            ],
            "resources": [
                "command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res",
                "command/danish-gcs/area-beijing/house-1059/floor-2/bin/2/res",
                "command/danish-gcs/area-beijing/house-1059/floor-3/bin/3/res",
                "command/danish-gcs/area-beijing/house-1060/floor-1/bin/4/res",
                "command/danish-gcs/area-beijing/house-1060/floor-2/bin/5/res",
                "command/danish-gcs/area-beijing/house-1060/floor-3/bin/6/res",
                "command/danish-gcs/area-beijing/house-1061/floor-1/bin/7/res",
                "command/danish-gcs/area-beijing/house-1061/floor-2/bin/8/res",
                "command/danish-gcs/area-beijing/house-1061/floor-3/bin/9/res",
                "command/danish-gcs/area-manchester/house-1440/floor-1/bin/10/res",
                "command/danish-gcs/area-manchester/house-1440/floor-2/bin/11/res",
                "command/danish-gcs/area-manchester/house-1440/floor-3/bin/12/res",
                "command/danish-gcs/area-manchester/house-1441/floor-1/bin/13/res",
                "command/danish-gcs/area-manchester/house-1441/floor-2/bin/14/res",
                "command/danish-gcs/area-manchester/house-1441/floor-3/bin/15/res"
            ]
        },
        "danish.GCS.TrashClassification:pubsub:2": {
```

For those who is interested to implement this in production, it can be observed that the inference request's payload does not contain any image. It is because according to the AWS official documentation, the maximum publish payload must be less than 128KB (Amazon Web Services, Inc., 2022a). It is assumed that the image was sent using other secure protocols such as HTTPS or SSL. In this assignment, the component is preloaded with the image for simplicity.

subscribe request				
Message size	The payload for every publish request can be no larger than 128 KB. AWS IoT Core rejects publish and connect requests larger than this size.	128 Kilobytes	128 Kilobytes	No
Outbound publish	Outbound publish requests count for every message that	20000	2000	Yes

7.4 Data Ingestion Component's Implementation Details

7.4.1 Component Dependencies

Data ingestion component is highly dependent on some of the components as shown below. As a result, the **DependencyType** is set to **HARD** in the component recipe so that the inference component restarts if any of the dependencies changes state.

f) danish.GCS.DLRInstallation: This component is required to manage the Conda environment and install the Python dependencies needed to run the component. Note that this component is shared with inference component.

g) MQTT bridge component: This component is required to enable communication between client devices and the core device through bridging MQTT topic mappings.

h) Stream manager component: This component is required to set up a stream manager client connection to export batches of data to the IoT Analytics channel.

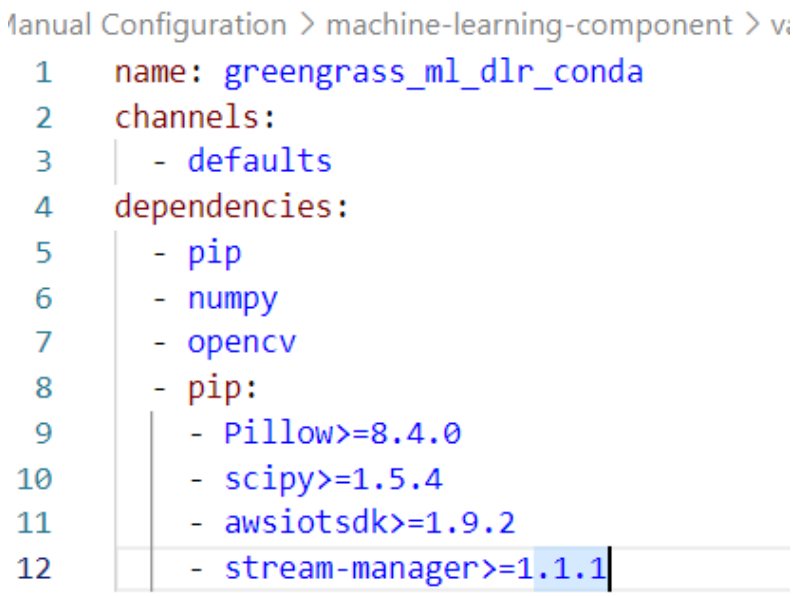
i) IP Detector component and Greengrass nucleus component: These components are required to start before the inference component can start.

```
"ComponentDependencies": {
    "danish.GCS.DLRInstallation": {
        "VersionRequirement": ">=1.0.0 <2.0.0",
        "DependencyType": "HARD"
    },
    "aws.greengrass.StreamManager": {
        "VersionRequirement": ">=2.0.14",
        "DependencyType": "HARD"
    },
    "aws.greengrass.clientdevices.IPDetector": {
        "VersionRequirement": ">=2.1.1",
        "DependencyType": "SOFT"
    },
    "aws.greengrass.clientdevices.mqtt.Bridge": {
        "VersionRequirement": ">=2.1.0",
        "DependencyType": "HARD"
    },
    "aws.greengrass.Nucleus": {
        "VersionRequirement": ">=2.0.0 <2.6.0",
        "DependencyType": "SOFT"
    }
},
```

7.4.2 Component's Artifact's Dependencies

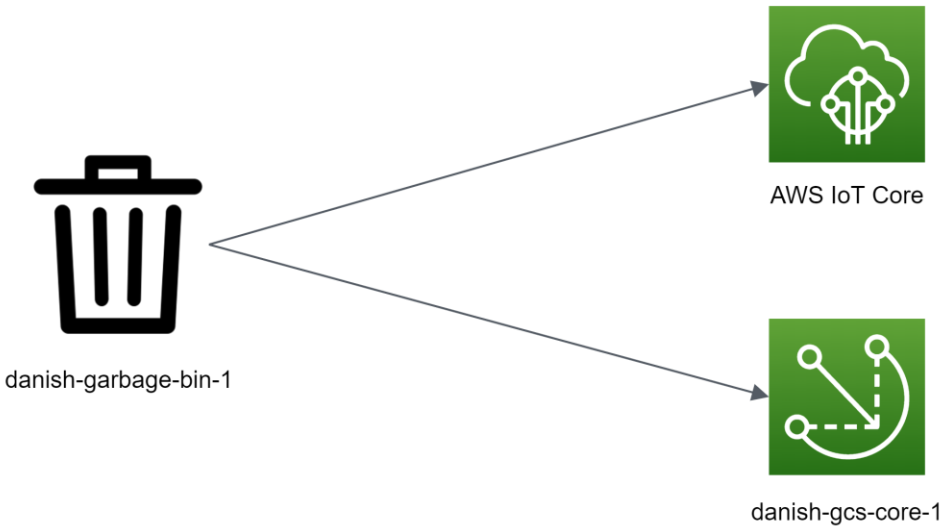
Below are the Python packages that are required in the data ingestion component:

a) pip: To install Python packages using pip.
b) awsiotsdk: Required to set up an IPC client connection to perform IPC operations such as subscribe to topics, receive messages from subscribed topics, publish messages to topics, and fetch component configuration's updates.
c) stream-manager: Required to set up a stream manager client connection to perform operations such as creating streams, appending message to the stream, and checking stream's upload status.



The file content above can be found in *s3_artifacts/variant.DLR.artifacts/environment.yaml.*

7.4.3 Sending Fill Level Telemetry



PUBLISH: data/danish-gcs/area-beijing/house-1059/floor-1/bin/1

TELEMETRY PAYLOAD:

```
{
   "thing_name": "danish-garbage-bin-1",
   "utc_timestamp": "1647819998",
   "fill_level": "0.2316"
}
```

1) To send the fill level telemetry, the client device connects to the local Moquette MQTT broker and publishes the fill level telemetry to the topic named **data/danish-gcs/area-beijing/house-1059/floor-1/bin/1**. Since the data ingestion component subscribes to this topic via local publish/subscribe messaging IPC service, the component will receive the fill level telemetry. Besides, the fill level telemetry will also be published to the AWS IoT Core. To enable such communication to happen, the following component recipes must be configured as shown below:

a) MQTT bridge component: A mapping is set up such that the MQTT bridge component listens for messages on the **data/danish-gcs/+/+/+/bin/+** topic filter from the local Moquette MQTT broker and publishes the same messages to the same topic in the Local publish/subscribe messaging IPC service and the AWS IoT Core MQTT broker.

```
"BinToIoTCoreMapping": {
    "topic": "data/danish-gcs/+/+/+/bin/+",
    "source": "LocalMqtt",
    "target": "IotCore"
},
"BinToStreamManagerClientComponentMapping": {
    "topic": "data/danish-gcs/+/+/+/bin/+",
    "source": "LocalMqtt",
    "target": "Pubsub"
},
```

b) Client device auth component: A policy named **danish-garbage-bin-1-policy** is set up such that the client device, in this case, **danish-garbage-bin-1**, can connect to the local Moquette MQTT broker and publish the fill level telemetry to the topic named **data/danish-gcs/area-beijing/house-1059/floor-1/bin/1**. Another policy named **danish-garbage-bin-dev-test-policy** is also set up such that **danish-garbage-bin-1** publish the fill level telemetry on behalf of other client devices for development and testing purposes. This is because only one virtual machine is set up for the client device and local Moquette MQTT broker expect only one device to use one device certificate.

```
"danish-garbage-bin-1-policy": {
    "AllowPublish": {
        "statementDescription": "Allow danish-garbage-bin-1 to publish fill level telemetry to the topic to
        be sent to various AWS cloud destinations.",
        "operations": [
            "mqtt:publish"
        ],
        "resources": [
            "mqtt:topic:data/danish-gcs/area-beijing/house-1059/floor-1/bin/1"
        ]
    },
```

```
"danish-garbage-bin-dev-test-policy": {
    "AllowPublish": {
        "statementDescription": "Allow danish-garbage-bin-1 to publish fill level telemetry on behalf of
        other client devices. As the name suggests, delete this policy in the production environment.",
        "operations": [
            "mqtt:publish"
        ],
        "resources": [
            "mqtt:topic:data/danish-gcs/area-beijing/house-1059/floor-2/bin/2",
            "mqtt:topic:data/danish-gcs/area-beijing/house-1059/floor-3/bin/3",
            "mqtt:topic:data/danish-gcs/area-beijing/house-1060/floor-1/bin/4",
            "mqtt:topic:data/danish-gcs/area-beijing/house-1060/floor-2/bin/5",
            "mqtt:topic:data/danish-gcs/area-beijing/house-1060/floor-3/bin/6",
            "mqtt:topic:data/danish-gcs/area-beijing/house-1061/floor-1/bin/7",
            "mqtt:topic:data/danish-gcs/area-beijing/house-1061/floor-2/bin/8",
            "mqtt:topic:data/danish-gcs/area-beijing/house-1061/floor-3/bin/9",
            "mqtt:topic:data/danish-gcs/area-manchester/house-1440/floor-1/bin/10",
            "mqtt:topic:data/danish-gcs/area-manchester/house-1440/floor-2/bin/11",
            "mqtt:topic:data/danish-gcs/area-manchester/house-1440/floor-3/bin/12",
            "mqtt:topic:data/danish-gcs/area-manchester/house-1441/floor-1/bin/13",
            "mqtt:topic:data/danish-gcs/area-manchester/house-1441/floor-2/bin/14",
            "mqtt:topic:data/danish-gcs/area-manchester/house-1441/floor-3/bin/15"
        ]
    }
}
```

c) Data ingestion component: In the component recipe, authorization policies must be defined to allow the data ingestion component to perform the IPC operations. In this case, a policy named **danish.GCS.TelemetryIngestion:pubsub:1** is defined to authorize the data ingestion component to subscribe to a list of topics as shown below to receive fill level telemetry.

```
"accessControl": {
    "aws.greengrass.ipc.pubsub": {
        "danish.GCS.TelemetryIngestion:pubsub:1": {
            "policyDescription": "Allow the component to receive telemetry from the topics published by the
            client devices.",
            "operations": [
                "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
                "data/danish-gcs/area-beijing/house-1059/floor-1/bin/1",
                "data/danish-gcs/area-beijing/house-1059/floor-2/bin/2",
                "data/danish-gcs/area-beijing/house-1059/floor-3/bin/3",
                "data/danish-gcs/area-beijing/house-1060/floor-1/bin/4",
                "data/danish-gcs/area-beijing/house-1060/floor-2/bin/5",
                "data/danish-gcs/area-beijing/house-1060/floor-3/bin/6",
                "data/danish-gcs/area-beijing/house-1061/floor-1/bin/7",
                "data/danish-gcs/area-beijing/house-1061/floor-2/bin/8",
                "data/danish-gcs/area-beijing/house-1061/floor-3/bin/9",
                "data/danish-gcs/area-manchester/house-1440/floor-1/bin/10",
                "data/danish-gcs/area-manchester/house-1440/floor-2/bin/11",
                "data/danish-gcs/area-manchester/house-1440/floor-3/bin/12",
                "data/danish-gcs/area-manchester/house-1441/floor-1/bin/13",
                "data/danish-gcs/area-manchester/house-1441/floor-2/bin/14",
                "data/danish-gcs/area-manchester/house-1441/floor-3/bin/15"
            ]
        }
    }
},
```

Section 8: References

Amazon Web Services, Inc., 2018. *batch-associate-client-device-with-core-device - AWS CLI 2.4.28 Command Reference*. [online] awscli.amazonaws.com. Available at: <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/greengrassv2/batch-associate-client-device-with-core-device.html> [Accessed 24 Mar. 2022].

Amazon Web Services, Inc., 2022a. *AWS IoT Core Endpoints and Quotas - AWS General Reference*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/general/latest/gr/iot-core.html#limits_iot> [Accessed 24 Mar. 2022].

Amazon Web Services, Inc., 2022b. *Compile and Deploy Models with Neo - Amazon SageMaker*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/sagemaker/latest/dg/neo.html> [Accessed 24 Mar. 2022].

Amazon Web Services, Inc., 2022c. *Install AWS IoT Greengrass Core Software with Automatic Resource Provisioning - AWS IoT Greengrass*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/greengrass/v2/developerguide/quick-installation.html> [Accessed 24 Mar. 2022].

Amazon Web Services, Inc., 2022d. *Install AWS IoT Greengrass Core Software with Manual Resource Provisioning - AWS IoT Greengrass*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/greengrass/v2/developerguide/manual-installation.html#create-token-exchange-role> [Accessed 24 Mar. 2022].

Amazon Web Services, Inc., 2022e. *MQTT Bridge - AWS IoT Greengrass*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/greengrass/v2/developerguide/mqtt-bridge-component.html> [Accessed 24 Mar. 2022].

Amazon Web Services, Inc., 2022f. *Prepare Model for Compilation - Amazon SageMaker*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/sagemaker/latest/dg/neo-compilation-preparing-model.html> [Accessed 24 Mar. 2022].

Andrew B, 2013. *Debian - Sudo Group Member Privileges in the Sudoers File*. [online] Server Fault. Available at: <https://serverfault.com/a/488646> [Accessed 24 Mar. 2022].

Hoffman, C., 2015. *Ubuntu Just Switched to systemd, the Project Sparking Controversy Throughout Linux*. [online] PCWorld. Available at: <https://www.pcworld.com/article/432370/ubuntu-just-switched-to-systemd-the-project-sparking-controversy-throughout-linux.html> [Accessed 24 Mar. 2022].

Jain, S., 2019. *What Is the Meaning and Impact of /run/user/0 and /run/user/1000 Directories in Linux(Fedora) ? | New Generation Enterprise Linux*. [online] New Generation Enterprise Linux. Available at: <https://ngelinux.com/what-is-the-meaning-and-impact-of-run-user-0-and-run-user-1000-directories-in-linuxfedora/> [Accessed 24 Mar. 2022].

Kerrisk, M., 2016. *tmpfs(5) - Linux Manual Page*. [online] man7.org. Available at: <https://man7.org/linux/man-pages/man5/tmpfs.5.html> [Accessed 24 Mar. 2022].

Oracle Corporation, 2022. *Chapter 6: Virtual Networking*. [online] www.virtualbox.org. Available at: <https://www.virtualbox.org/manual/ch06.html#network_hostonly> [Accessed 24 Mar. 2022].

phemmer, 2014. *Linux - What Is This Folder /run/user/1000?* [online] Unix & Linux Stack Exchange. Available at: <https://unix.stackexchange.com/a/162911> [Accessed 24 Mar. 2022].