



UNIVERSITI TUNKU ABDUL RAHMAN

ACADEMIC YEAR 2022/2023

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY (PERAK)

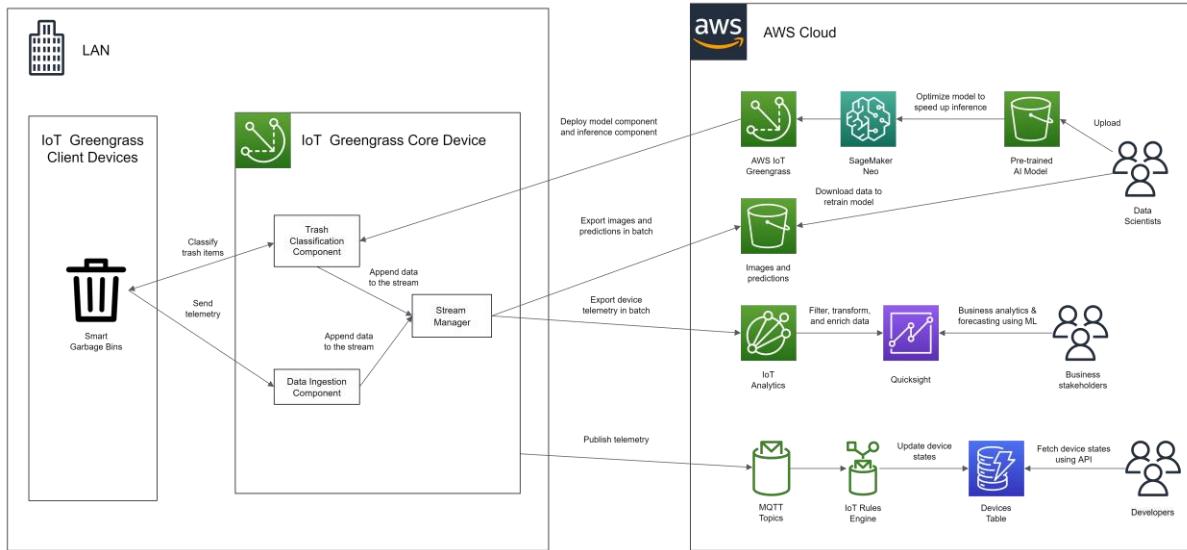
UCCD3113 DISTRIBUTED COMPUTER SYSTEMS

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.....	7
1.1 AWS Console.....	7
1.2 Setup AWS CLI .....	7
1.3 Setup Virtual Machines.....	8
Section 2: Provisioning AWS Cloud Resource .....	11
2.1 IAM Roles and Policies .....	11
2.1.1 Service Role #1: DanishGBSGreengrassV2TokenExchangeRole .....	12
2.1.2 Service Role #2: DanishGCSSageMakerIAMRole .....	14
2.1.3 Service Role #3: DanishGCSBulkRegistrationAccessS3Role .....	15
2.1.4 Service Role #4: DanishGCSDescribeThingRole .....	16
2.2 Amazon DynamoDB.....	17
2.3 Amazon S3.....	18
2.4 Amazon SageMaker Neo Compilation.....	19
2.5 AWS IoT Core Resources.....	23
2.6 AWS IoT Rules Engine .....	24
Section 3: Client Device Setup .....	28
3.1 Register Client Devices in Bulk.....	30
3.2 Establish local connection between client devices and core devices.....	33
Section 4: Greengrass Core Device Setup .....	37
4.1 Create a Greengrass Service Role.....	37
4.2 Check Device Requirements.....	39
4.2.1 Linux system uses GNU C Library (glibc) version 2.25 or greater.....	40
4.2.2 Java Runtime Environment (JRE) version 8 or greater is installed.....	40
4.2.3 User must have the permission to run sudo with any user and any group.....	41
4.2.4 The /tmp directory must be mounted with exec permissions. ....	42
4.2.5 Certain shell commands must be available.....	43

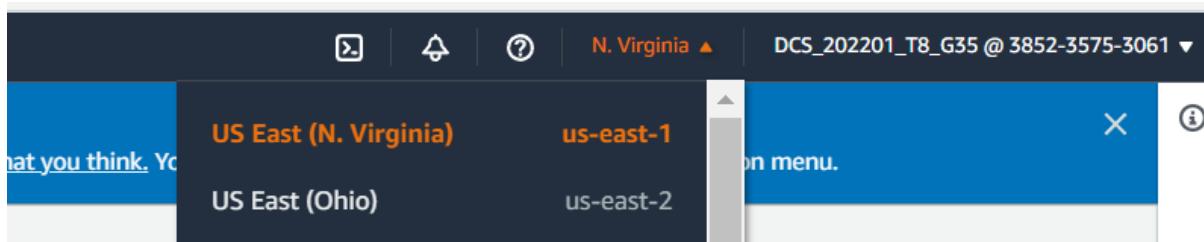
4.3 Install Greengrass Core Software with Manual Provisioning.....	44
4.4 Install Greengrass Core Software with Automatic Provisioning.....	47
4.5 Validate Greengrass Core Software Installation.....	48
4.6 Prepare Greengrass Component's Artifacts for Cloud Deployment.....	49
4.7 Deploy AWS IoT Greengrass Component .....	50
4.8 Data Ingestion Component Validation.....	55
4.8.1 Validate Command's Logs from Garbage Bin/Client Device .....	56
4.8.2 Validate Component's Logs from Core Device.....	57
4.8.3 Validate Results from AWS Management Console.....	59
4.9 Inference Component Validation .....	61
4.9.1 Validate Command's Logs from Garbage Bin/Client Device .....	62
4.9.2 Validate Component's Logs from Core Device.....	63
4.9.3 Validate Results from AWS Management Console.....	65
4.10 Shutdown Greengrass System Service.....	66
Section 5: AWS IoT Analytics .....	67
5.1 Create Channel.....	67
5.2 Create Data Store .....	68
5.3 Create Pipeline .....	69
5.4 Create Dataset .....	72
Section 6: Amazon QuickSight.....	76
Section 7: Developer Guide .....	80
7.1 Local Deployment of Greengrass Component.....	80
7.2 Communication between Client Devices with Core Device and AWS IoT Core .....	82
7.3 Inference Component's Implementation Details .....	83
7.3.1 Component Dependencies .....	83
7.3.2 Component's Artifact's Dependencies .....	84
7.3.3 Sending Inference Request .....	85

7.3.4 Receiving Inference Response .....	87
7.4 Data Ingestion Component's Implementation Details .....	90
7.4.1 Component Dependencies .....	90
7.4.2 Component's Artifact's Dependencies .....	91
7.4.3 Sending Fill Level Telemetry .....	92
Section 8: References.....	95

## 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.

## Summary

A screenshot of the AWS IAM User Summary page. The 'Access Advisor' tab is selected. On the left, there's a 'Sign-in credentials' section with a 'Summary' table and a 'Console password' section. Below that is an 'Access keys' section with a note about using them for programmatic calls. A modal window titled 'Create access key' is overlaid, containing a warning about posting secret access keys publicly and a success message indicating the key can be viewed or downloaded once. The access key ID is AKIAV[REDACTED] and the secret access key is kXwmspBbZw[REDACTED].

- 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

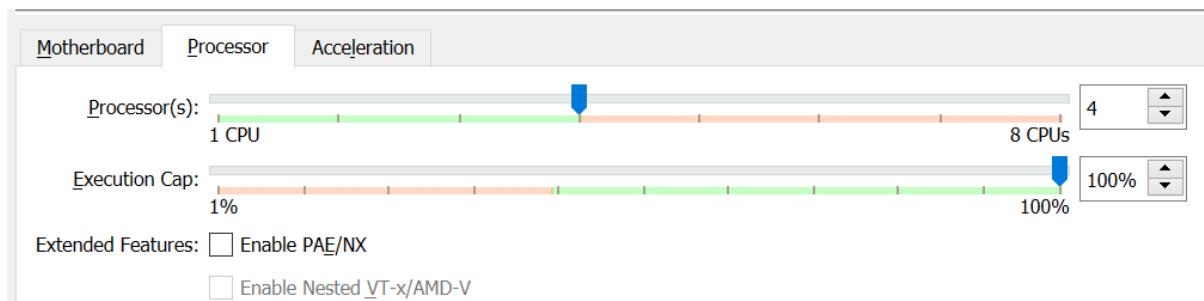
```
C:\Users\User>aws configure
AWS Access Key ID [*****IKBN]: AKI[REDACTED]
AWS Secret Access Key [*****v4ZA]: kXwmspBbZw[REDACTED] QJ/N
Default region name [ap-southeast-1]: us-east-1
Default output format [json]: json
```

## 1.3 Setup Virtual Machines

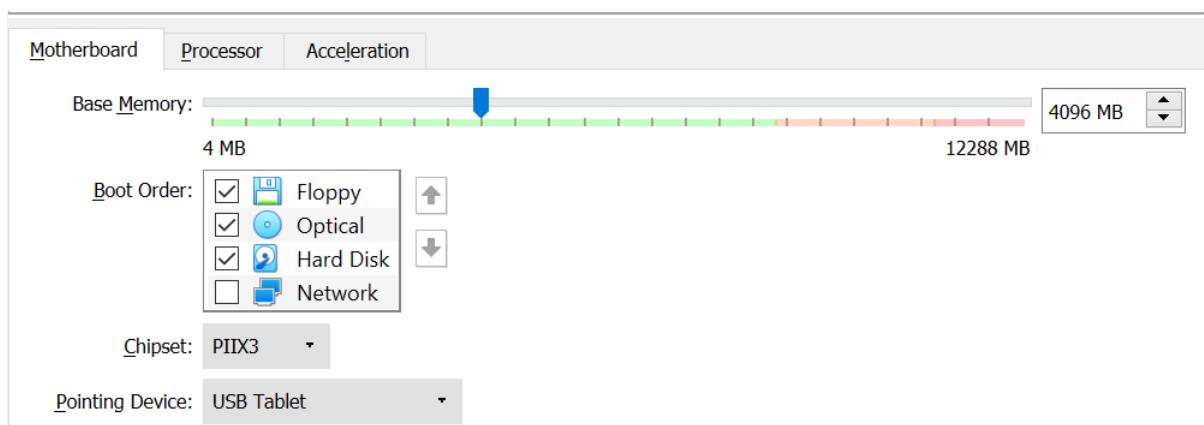
- 1) There is no hardware to implement the system.
- 2) The virtualization software used is Oracle VM VirtualBox Version 6.1.32. The virtual machines for core device and client device use 64-bit Ubuntu 18.04.6. The ISO image can be downloaded from <https://releases.ubuntu.com/18.04>. Feel free to install 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. **danh-garbage-bin-1@danh-garbage-bin-1** is referring to the client device running on Ubuntu. **jhengkhinyap@danh-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

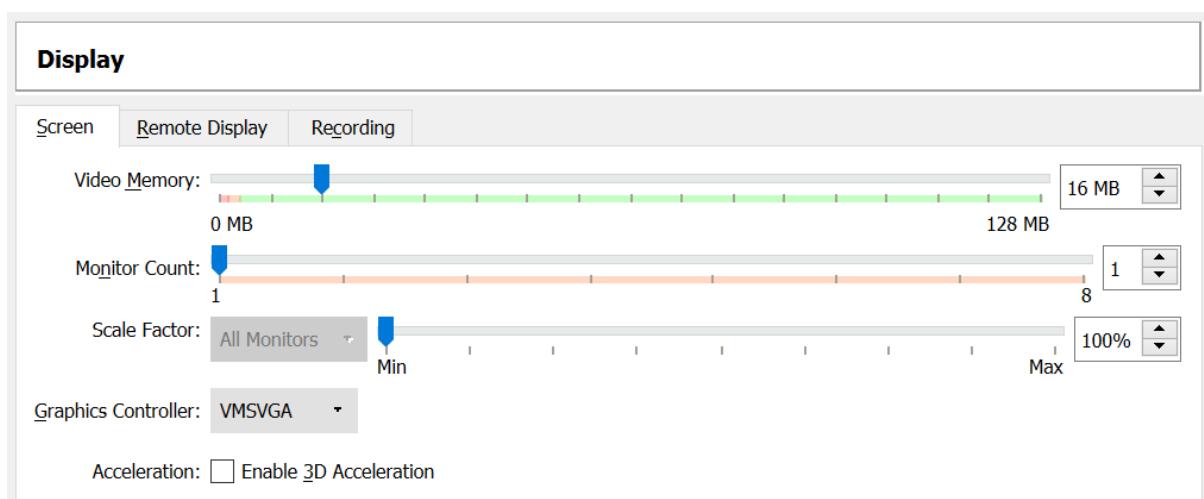
- 1) The optimal resource requirements of the Ubuntu 64-bit virtual machine for core device and client device are shown in the image form. Adjust the requirement accordingly based on your host OS and hardware specifications.
- 2) The minimum required processor is one.



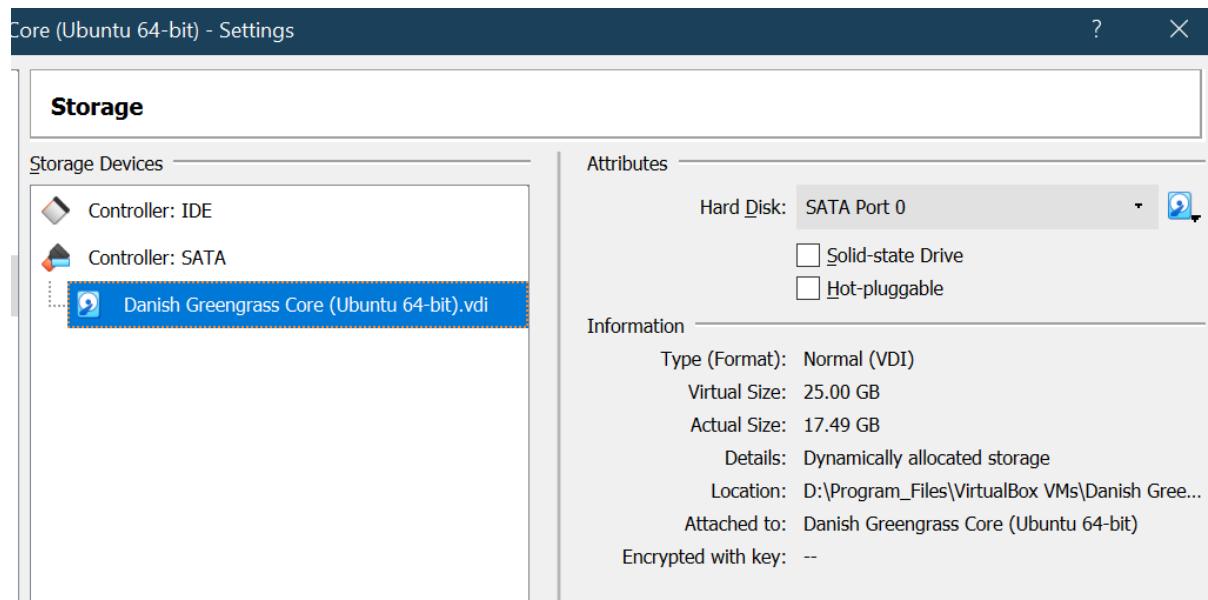
- 3) The minimum required RAM is 2048MiB.



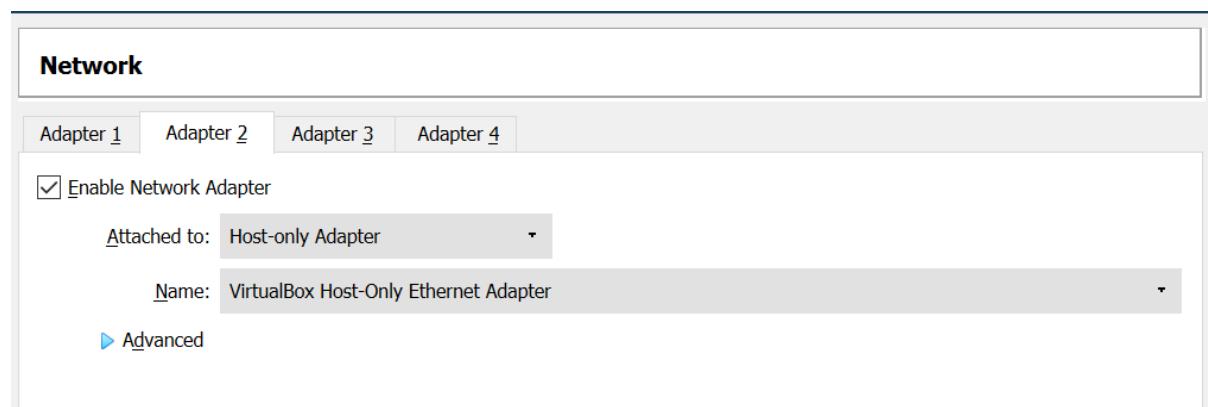
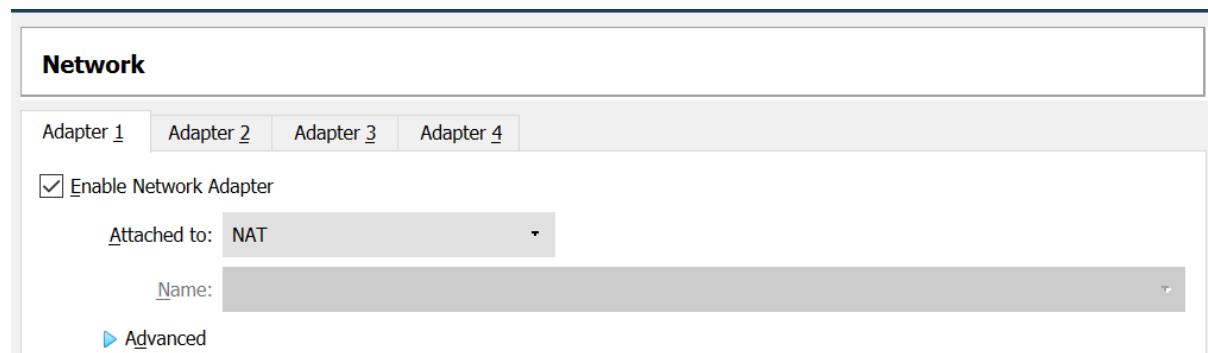
- 4) Then minimum required video memory is 16MiB.



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**.

The screenshot shows the AWS IAM Policy JSON editor. The tabs at the top are "Visual editor" and "JSON", with "JSON" being selected. On the right, there is a button labeled "Import managed policy". The code area contains the following JSON:

```
2 "Version": "2012-10-17",
3 "Statement": [
4     {
5         "Effect": "Allow",
6         "Action": [
7             "logs:CreateLogGroup",
8             "logs:CreateLogStream",
9             "logs:PutLogEvents",
10            "logs:DescribeLogStreams",
11            "sns:Publish"
12        ]
13    }
14]
```

- 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**.

### Select trusted entity

#### Trusted entity type

The screenshot shows the "Select trusted entity" step in the IAM Role creation wizard. It lists four options:

- AWS service: Allows AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account: Allows entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- SAML 2.0 federation: Allows users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- Custom trust policy: Allows creating a custom trust policy to enable others to perform actions in this account.

#### Custom trust policy

Create a custom trust policy to enable others to perform actions in this account.

The screenshot shows the "Custom trust policy" editor. The code area contains the following JSON template:

```
1 "Version": "2012-10-17",
2 "Statement": [
3     {
4         "Effect": "Allow",
5         "Principal": {
6             "Service": "credentials.iot.amazonaws.com"
7         },
8         "Action": "sts:AssumeRole"
9     }
10 ]
11 ]
12 ]
```

- 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**.

**Permissions policies (Selected 1/741)**  
Choose one or more policies to attach to your new role.

Filter policies by property or policy name and press enter

"DanishGBSGreengrassV2TokenExchangeRoleAccess" X | Clear filters

Policy name	Type	Description
DanishGBSGreengrassV2TokenExchangeRoleAccess	Customer manag...	

► **Set permissions boundary - optional**

Set a permissions boundary to control the maximum permissions this role can have. This is not a common setting, but you can use it to delegate permission management to others.

- 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": ["s3GetObject"],  
      "Effect": "Allow",  
      "Resource": [  
        "arn:aws:s3::::danish-gcs-model-artifacts-bucket/v*/pytorch/*"  
      ]  
    },  
    {  
      "Action": ["s3PutObject"],  
      "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: **AWSIoTTThingsRegistration**

- 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**

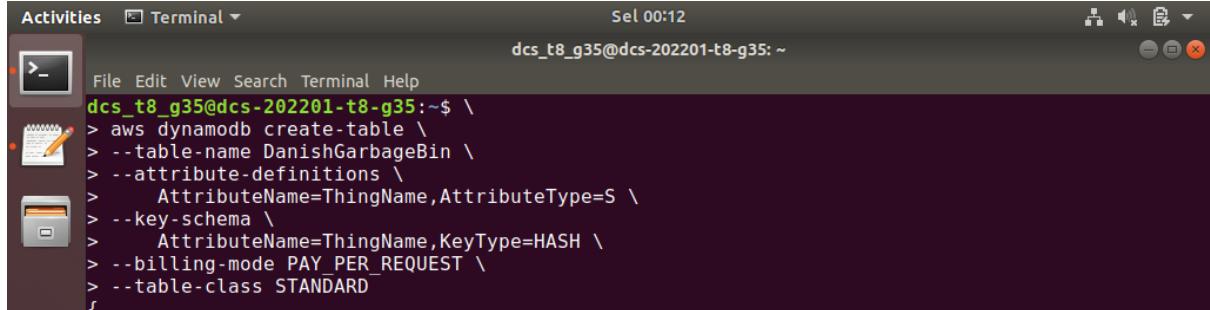
- Permission policy:

```
{  
  "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

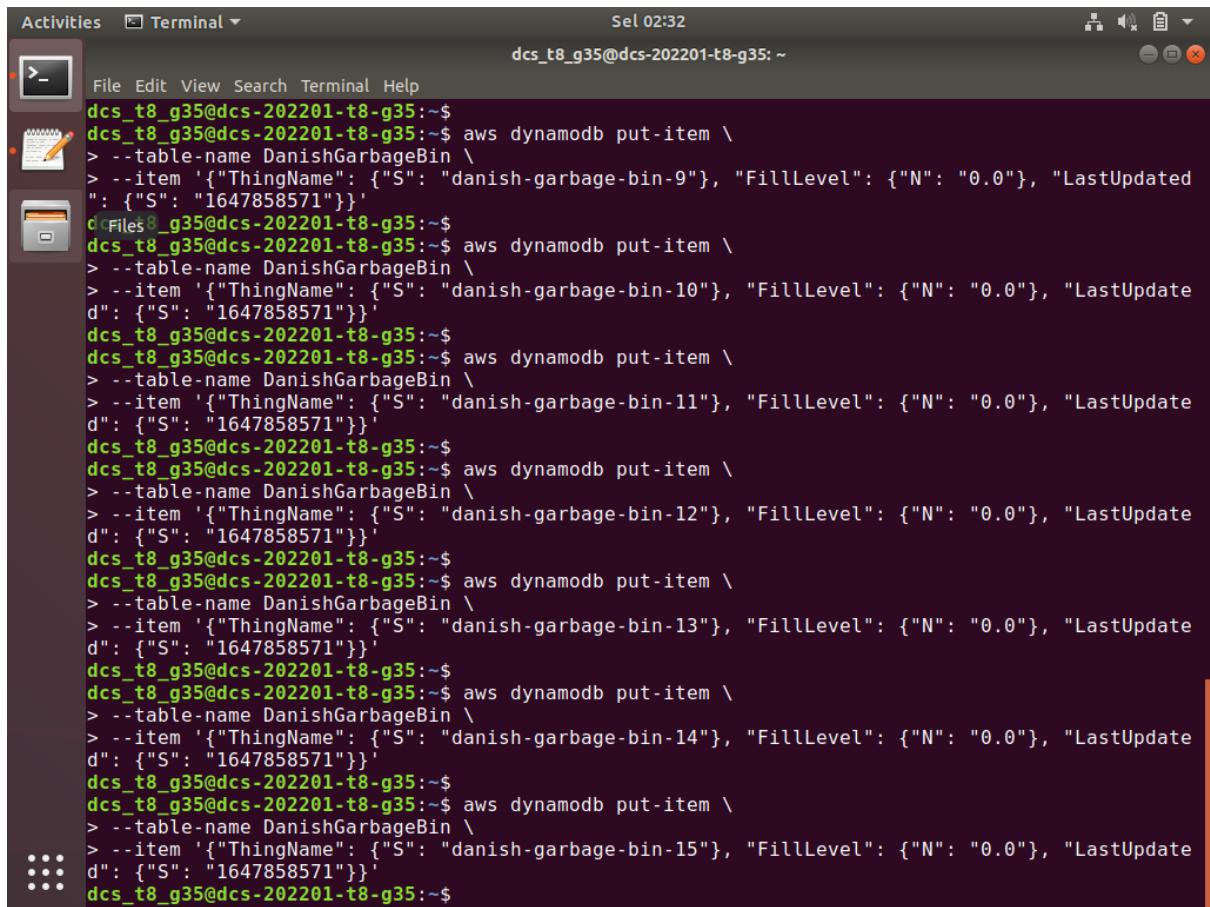
Note: The commands can be found in *aws\_cli/dynamodb\_commands.txt*.

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



```
Activities Terminal ▾ Sel 00:12
dcs_t8_g35@dcs-202201-t8-g35:~$ 
File Edit View Search Terminal Help
dcs_t8_g35@dcs-202201-t8-g35:~$ \
> aws dynamodb create-table \
> --table-name DanishGarbageBin \
> --attribute-definitions \
>   AttributeName=ThingName,AttributeType=S \
> --key-schema \
>  AttributeName=ThingName,KeyType=HASH \
> --billing-mode PAY_PER_REQUEST \
> --table-class STANDARD
{
```

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



```
Activities Terminal ▾ Sel 02:32
dcs_t8_g35@dcs-202201-t8-g35:~$ 
File Edit View Search Terminal Help
dcs_t8_g35@dcs-202201-t8-g35:~$ aws dynamodb put-item \
> --table-name DanishGarbageBin \
> --item '{"ThingName": {"S": "danish-garbage-bin-9"}, "FillLevel": {"N": "0.0"}, "LastUpdated": {"S": "1647858571"}}'
dcs_t8_g35@dcs-202201-t8-g35:~$ aws dynamodb put-item \
> --table-name DanishGarbageBin \
> --item '{"ThingName": {"S": "danish-garbage-bin-10"}, "FillLevel": {"N": "0.0"}, "LastUpdate": {"S": "1647858571"}}'
dcs_t8_g35@dcs-202201-t8-g35:~$ aws dynamodb put-item \
> --table-name DanishGarbageBin \
> --item '{"ThingName": {"S": "danish-garbage-bin-11"}, "FillLevel": {"N": "0.0"}, "LastUpdate": {"S": "1647858571"}}'
dcs_t8_g35@dcs-202201-t8-g35:~$ aws dynamodb put-item \
> --table-name DanishGarbageBin \
> --item '{"ThingName": {"S": "danish-garbage-bin-12"}, "FillLevel": {"N": "0.0"}, "LastUpdate": {"S": "1647858571"}}'
dcs_t8_g35@dcs-202201-t8-g35:~$ aws dynamodb put-item \
> --table-name DanishGarbageBin \
> --item '{"ThingName": {"S": "danish-garbage-bin-13"}, "FillLevel": {"N": "0.0"}, "LastUpdate": {"S": "1647858571"}}'
dcs_t8_g35@dcs-202201-t8-g35:~$ aws dynamodb put-item \
> --table-name DanishGarbageBin \
> --item '{"ThingName": {"S": "danish-garbage-bin-14"}, "FillLevel": {"N": "0.0"}, "LastUpdate": {"S": "1647858571"}}'
dcs_t8_g35@dcs-202201-t8-g35:~$ aws dynamodb put-item \
> --table-name DanishGarbageBin \
> --item '{"ThingName": {"S": "danish-garbage-bin-15"}, "FillLevel": {"N": "0.0"}, "LastUpdate": {"S": "1647858571"}}'
dcs_t8_g35@dcs-202201-t8-g35:~$
```

## 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**.

The screenshot shows the 'Create bucket' wizard in the Amazon S3 console. The 'General configuration' step is selected. The 'Bucket name' field contains 'danish-gcs-bulk-reg-params-bucket'. The 'AWS Region' dropdown is set to 'US East (N. Virginia) us-east-1'. Below these fields is a section for copying settings from an existing bucket, with a 'Choose bucket' button.

- 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.

The screenshot shows the Amazon S3 console interface. The path is: Amazon S3 > Buckets > danish-gcs-model-artifacts-bucket > v1/ > pytorch/. The 'pytorch/' folder contains one object: 'danish\_trash\_cf\_model.tar.gz'. The object details are as follows:

Name	Type	Last modified	Size	Storage class
danic_trash_cf_model.tar.gz	gz	March 11, 2022, 20:36:51 (UTC+08:00)	39.7 MB	Standard

- 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.

## Create compilation job

### Job settings

The settings define the job, the credentials for accessing Amazon S3, and set constraints on the cost of running the job.

#### Job name

danish-gcs-model-v1-compilation-job

The name must be between 1 and 63 characters in length and unique in your AWS account and AWS region. Valid characters and a-z, A-Z, 0-9, and hyphen (-).

#### IAM role

Amazon SageMaker requires permissions to call other services on your behalf. Choose a role or let us create a role that has the [AmazonSageMakerFullAccess](#) IAM policy attached.

DanishGCSSageMakerIAMRole

- 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).

### Input configuration

Amazon SageMaker needs to know where model artifacts are stored, what the shape of the data matrix is, and which machine learning framework to use. [Learn more](#)

Model artifacts

Enter the input configuration directly for your model artifacts.

Model version

Use a versioned model from a model package group.

Location of model artifacts

Amazon SageMaker needs the path to the model artifacts in Amazon S3. To find the path, look in your Amazon S3 directories.

s3://danish-gcs-model-artifacts-bucket/v1/pytorch/danish\_trash\_cf\_model.tar.gz

To find a path, [go to Amazon S3](#)

Data input configuration

Amazon SageMaker needs to know what the shape of the data matrix is.

{"data": [1,3,224,224]}

Machine learning framework

Choose the machine learning framework that your model was trained in.

PyTorch

Framework version

Choose the machine learning framework version that your model was trained in.

1.8

- 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
```

```

dcs_t8_g35@dcs-202201-t8-g35:~$ uname -a
Linux dcs-202201-t8-g35 5.4.0-84-generic #94~18.04.1-Ubuntu SMP Thu Aug 26 23:17:46
UTC 2021 x86_64 x86_64 x86_64 GNU/Linux

dcs_t8_g35@dcs-202201-t8-g35:~$ cat /proc/cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 142
model name     : Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
stepping        : 11
microcode      : 0xffffffff
cpu MHz        : 1800.005
cache size     : 6144 KB
physical id    : 0
siblings        : 2
core id         : 0
cpu cores      : 2
apicid          : 0

```

- 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.

**Output configuration**  
Amazon SageMaker needs to know where to store the modules compiled with this job. [Learn more](#)

**Target device**  
Choose the target device or the machine learning instance that you want to run your model on after the compilation has completed.

**Target platform**  
Control the target platform that you want your model to run on, such as OS, architecture, and accelerators.

**OS**  
Specify a target platform OS.

**Arch**  
Specify a target platform architecture.

**Accelerator - optional**  
Specify a target platform accelerator.

**Compiler options - optional**  
Specify additional parameters for compiler options in JSON format.

**S3 Output location**  
Amazon SageMaker needs the path to the S3 bucket or folder where you want to store the compiled module.

To find a path, [go to Amazon S3](#)

**Encryption key - optional**  
Encrypt your data. Choose an existing KMS key or enter a key's ARN.

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

Compilation jobs					
		C	Actions ▾	Create model	Create compilation job
<input type="text"/> Search compilation jobs					
	Name	Status	Target	Duration	Creation time
<input type="radio"/>	danish-gcs-model-v1-compilation-job-2	<span>✓ COMPLETED</span>	linux-x86_64	4 minutes	Mar 11, 2022 13:43 UTC
<input type="radio"/>	danish-gcs-model-v1-compilation-job	<span>✗ FAILED</span>	linux-x86_64	3 minutes	Mar 11, 2022 12:50 UTC

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.

Amazon S3 > Buckets > danish-gcs-model-artifacts-bucket > v1/ > neo-compiled/

**neo-compiled/**

**Objects** | **Properties**

**Objects (2)**

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

[Find objects by prefix](#) < 1 > ⚙️

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">danish_trash_cf_model-LINUX_X86_64.zip</a>	zip	March 12, 2022, 23:44:05 (UTC+08:00)	40.7 MB	Standard
<input type="checkbox"/>	<a href="#">danish_trash_cf_model-LINUX_X86_64.tar.gz</a>	gz	March 12, 2022, 09:38:34 (UTC+08:00)	40.7 MB	Standard

## 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
```

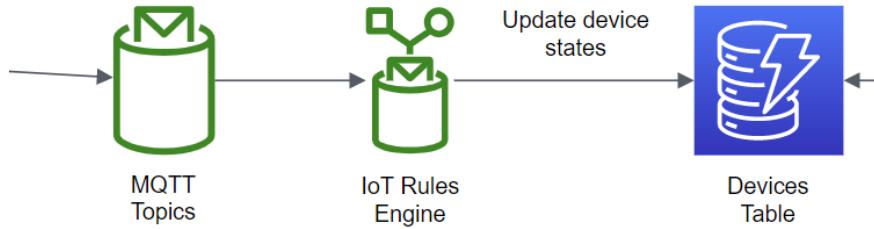
- 3) Execute the commands below to create thing policy for the client device. The policy can be found in *aws\_cli/DanishGCSBinThingPolicy.json*.

```
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**.

Create a rule to evaluate messages sent by your things and specify what to do when a message is received (for example, write data to a DynamoDB table or invoke a Lambda function).

Name

UpdateDanishBinFillLevelRule

Description

Forward the bin's fill level to the Dynamo DB, which then can be accessed by web and mobile applications.

- 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/+'

### Rule query statement

Indicate the source of the messages you want to process with this rule.

#### Using SQL version

2016-03-23

#### Rule query statement

SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see [AWS IoT SQL Reference](#).

```
1  SELECT
2    thing_name AS ThingName,
3    utc_timestamp AS LastUpdated,
4    fill_level AS FillLevel
5  FROM 'data/danish-gcs/+/+/bin/+'
```

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

Select an action

Select an action.

-  Insert a message into a DynamoDB table  
DYNAMODB
-  Split message into multiple columns of a DynamoDB table (DynamoDBv2)  
DYNAMODBV2
-  Send a message to a Lambda function  
LAMBDA

- 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 **DanishGCSDynoDBWriteAccessRole**. Select **Add action**.

Configure action

 Split message into multiple columns of a DynamoDB table (DynamoDBv2)  
DYNAMODBV2

The DynamoDBv2 action allows you to write all or part of an MQTT message to a DynamoDB table. Each attribute in the payload is written to a separate column in the DynamoDB database. Messages processed by this action must be in the JSON format.

\*Table name

DanishGarbageBin

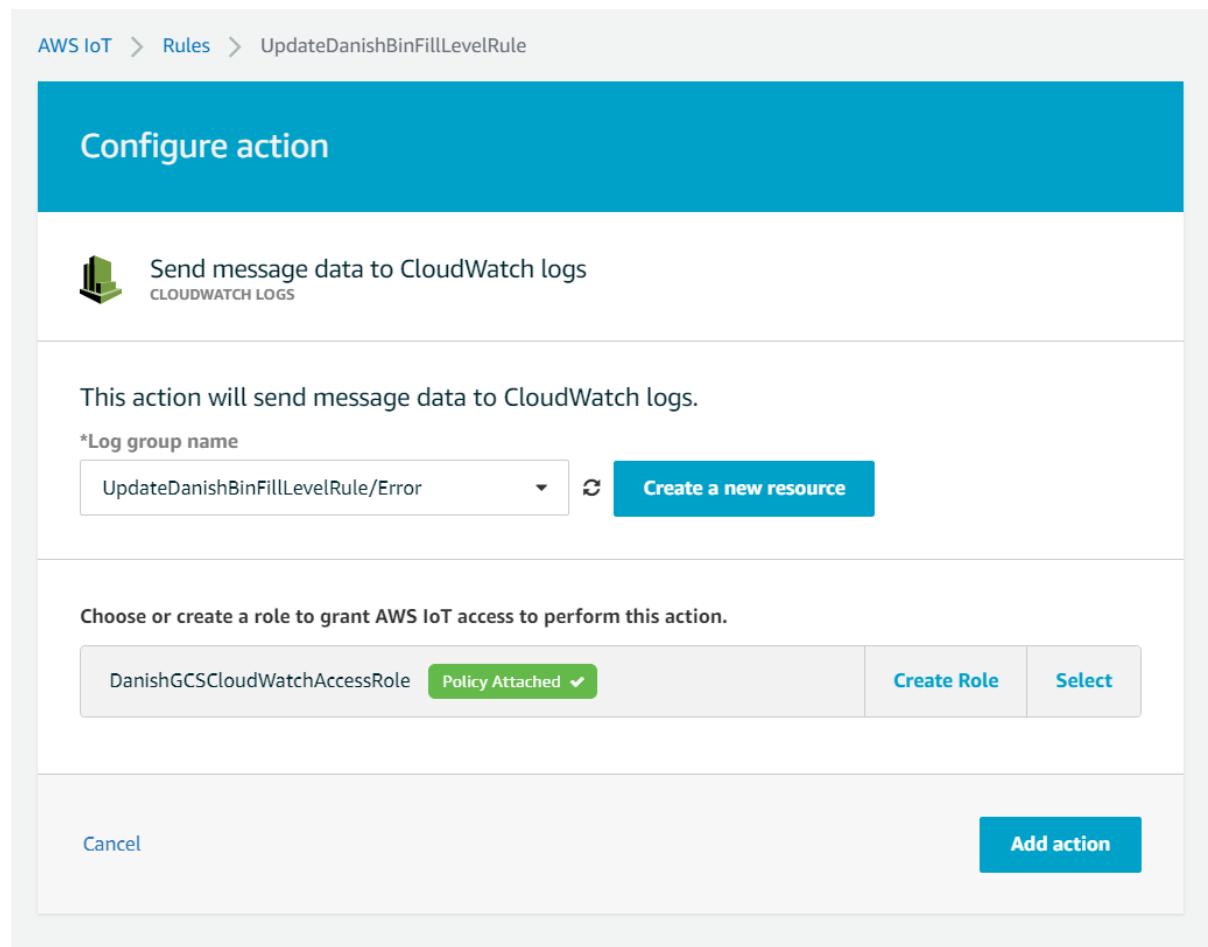
Choose or create a role to grant AWS IoT access to perform this action.

DanishGCSDynoDBWriteAccessRole

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

-  Send message data to CloudWatch metric  
CLOUDWATCH METRICS
-  Change the state of a CloudWatch alarm  
CLOUDWATCH ALARMS
-  Send message data to CloudWatch logs  
CLOUDWATCH LOGS
-  Send a message to the Amazon OpenSearch Service (successor to Amazon Elasticsearch Service)  
AMAZON OPENSEARCH SERVICE (SUCCESSOR TO AMAZON ELASTICSEARCH SERVICE)
-  Send a message to a Salesforce IoT Input Stream

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**.



AWS IoT > Rules > UpdateDanishBinFillLevelRule

### Configure action

 Send message data to CloudWatch logs  
CLOUDWATCH LOGS

This action will send message data to CloudWatch logs.

\*Log group name

UpdateDanishBinFillLevelRule/Error

Choose or create a role to grant AWS IoT access to perform this action.

DanishGCSCloudWatchAccessRole	Policy Attached ✓	<a href="#">Create Role</a>	<a href="#">Select</a>
-------------------------------	-------------------	-----------------------------	------------------------

[Cancel](#) [Add action](#)

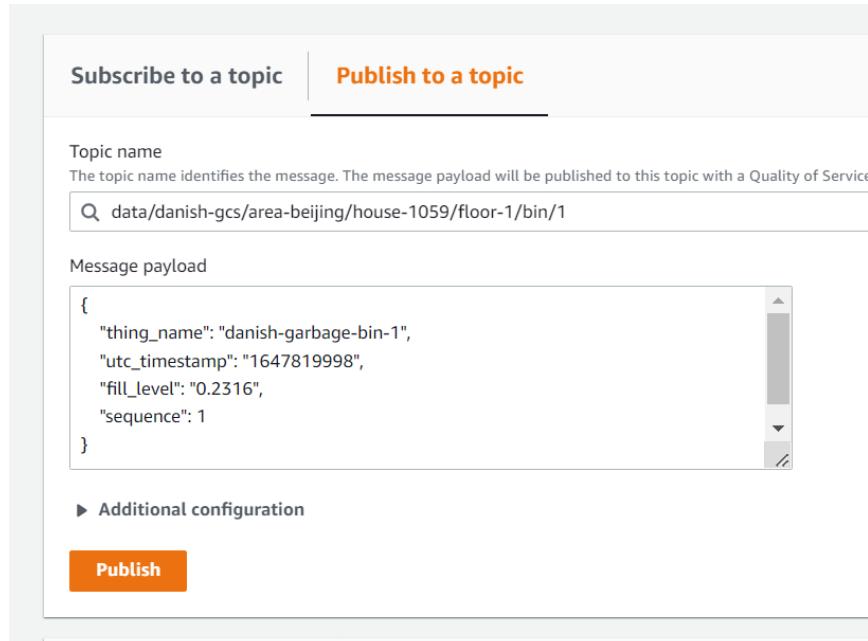
7) Finally, select **Create rule**.

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  
}
```

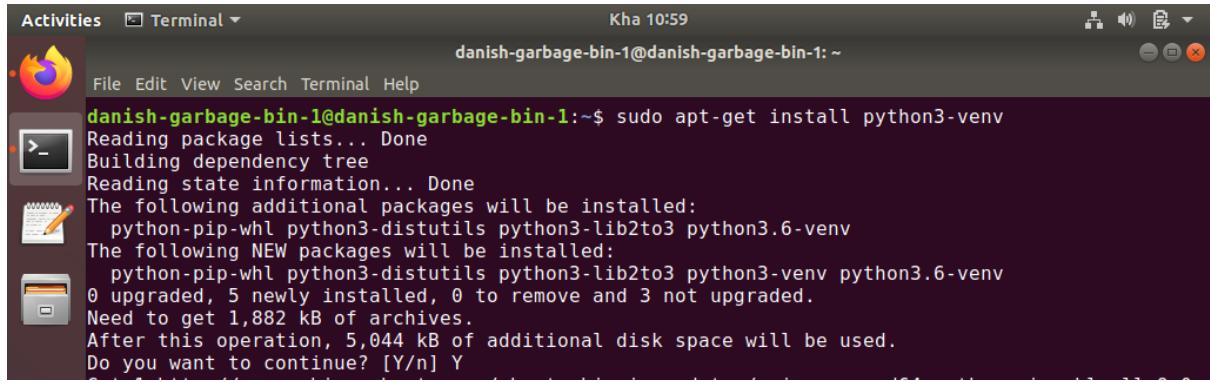


9) In the **DynamoDB Console** navigation menu, select **Tables > DanishGarbageBin > Explore table items**. Check that the corresponding item has been updated. If not, check the CloudWatch logs for any error.

Completed Read capacity units consumed: 2			
Items returned (15)			
	ThingName	FillLevel	LastUpdated
	danish-garbage-bin-1	0.2316	1647819998

### 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.

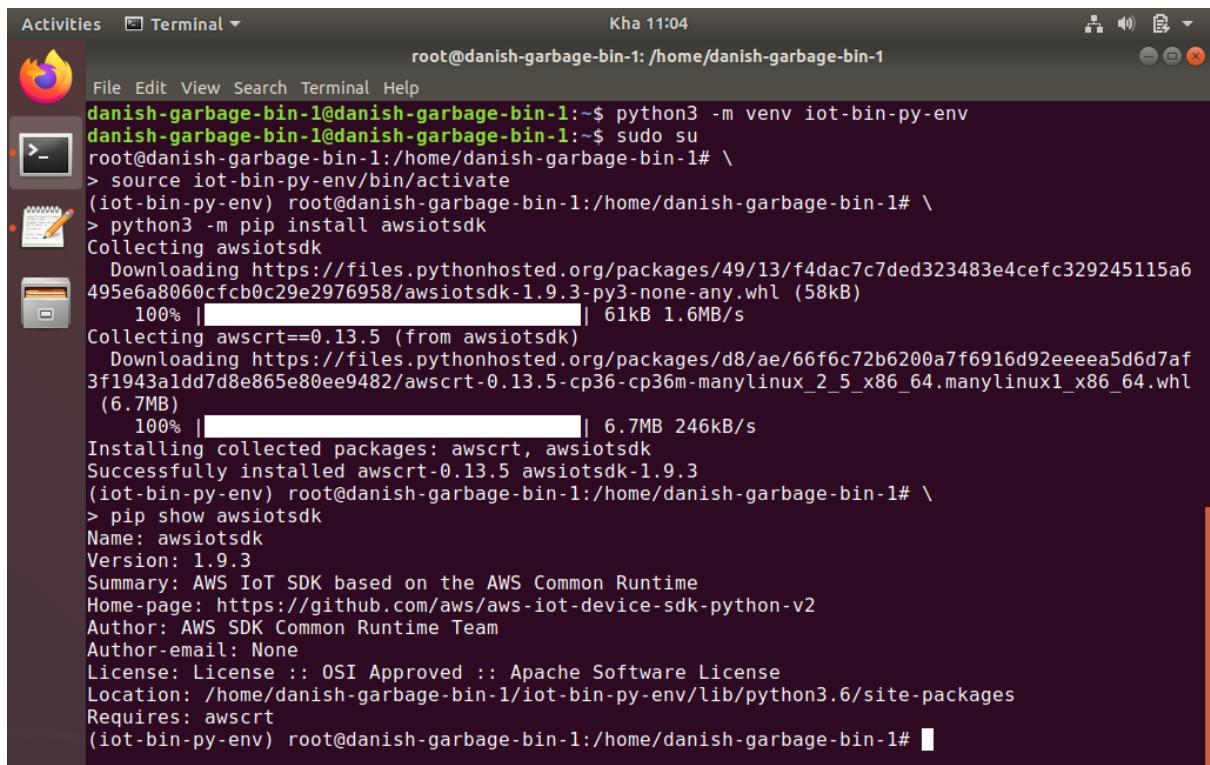


A screenshot of a Linux desktop environment showing a terminal window titled "Terminal". The terminal window has a dark background and displays the command "sudo apt-get install python3-venv" and its output. The output shows the package lists being read, dependency tree building, state information, and the installation of several packages including python-pip-whl, python3-distutils, python3-lib2to3, python3.6-venv, and python3-venv. It also indicates 0 upgraded, 5 newly installed, and 3 not upgraded packages. The total size of the archives is 1,882 kB and 5,044 kB of additional disk space will be used. A confirmation prompt "Do you want to continue? [Y/n] Y" is shown at the end.

```
danish-garbage-bin-1@danish-garbage-bin-1:~$ sudo apt-get install python3-venv
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  python-pip-whl python3-distutils python3-lib2to3 python3.6-venv
The following NEW packages will be installed:
  python-pip-whl python3-distutils python3-lib2to3 python3-venv python3.6-venv
0 upgraded, 5 newly installed, 0 to remove and 3 not upgraded.
Need to get 1,882 kB of archives.
After this operation, 5,044 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

- 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
```

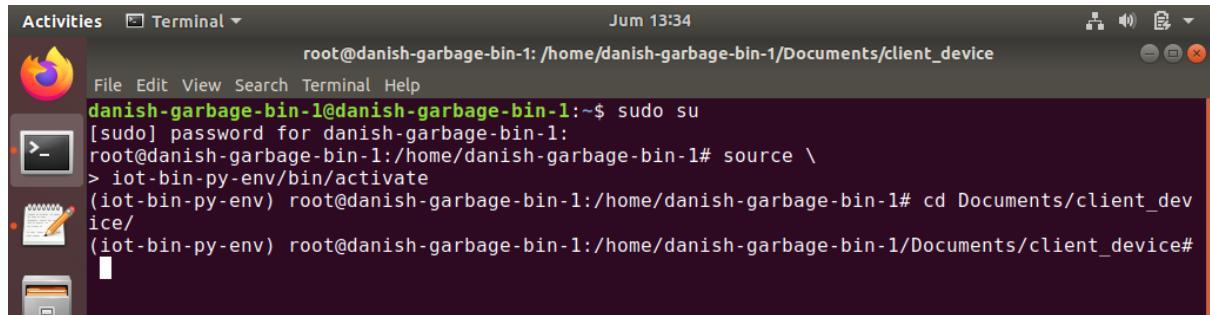


A screenshot of a Linux desktop environment showing a terminal window titled "Terminal". The terminal window has a dark background and displays the command "python3 -m venv iot-bin-py-env" and its output. The output shows the user switching to root using "sudo su", activating the virtual environment with "source iot-bin-py-env/bin/activate", and then installing the awsiotsdk package using "python3 -m pip install awsiotsdk". The process includes package collection, download of awscrt and awsiotsdk wheels, and successful installation. Finally, the user runs "pip show awsiotsdk" to view the package details.

```
root@danish-garbage-bin-1:/home/danish-garbage-bin-1#
root@danish-garbage-bin-1:~$ python3 -m venv iot-bin-py-env
root@danish-garbage-bin-1:~$ sudo su
root@danish-garbage-bin-1:/home/danish-garbage-bin-1# \
> source iot-bin-py-env/bin/activate
(iot-bin-py-env) root@danish-garbage-bin-1:/home/danish-garbage-bin-1# \
> python3 -m pip install awsiotsdk
Collecting awsiotsdk
  Downloading https://files.pythonhosted.org/packages/49/13/f4dac7c7ded323483e4cefc329245115a6
  495e6a8060cfcb0c29e2976958/awsiotsdk-1.9.3-py3-none-any.whl (58kB)
    100% |██████████| 61kB 1.6MB/s
Collecting awscrt==0.13.5 (from awsiotsdk)
  Downloading https://files.pythonhosted.org/packages/d8/ae/66f6c72b6200a7f6916d92eeeea5d6d7af
  3f1943a1dd7d8e865e80ee9482/awscrt-0.13.5-cp36-cp36m-manylinux_2_5_x86_64.manylinux1_x86_64.whl
  (6.7MB)
    100% |██████████| 6.7MB 246kB/s
Installing collected packages: awscrt, awsiotsdk
Successfully installed awscrt-0.13.5 awsiotsdk-1.9.3
(iot-bin-py-env) root@danish-garbage-bin-1:/home/danish-garbage-bin-1# \
> pip show awsiotsdk
Name: awsiotsdk
Version: 1.9.3
Summary: AWS IoT SDK based on the AWS Common Runtime
Home-page: https://github.com/aws/aws-iot-device-sdk-python-v2
Author: AWS SDK Common Runtime Team
Author-email: None
License: License :: OSI Approved :: Apache Software License
Location: /home/danish-garbage-bin-1/iot-bin-py-env/lib/python3.6/site-packages
Requires: awscrt
(iot-bin-py-env) root@danish-garbage-bin-1:/home/danish-garbage-bin-1# 
```

3) Run the command below to run the Python files located in *client\_device* directory.

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



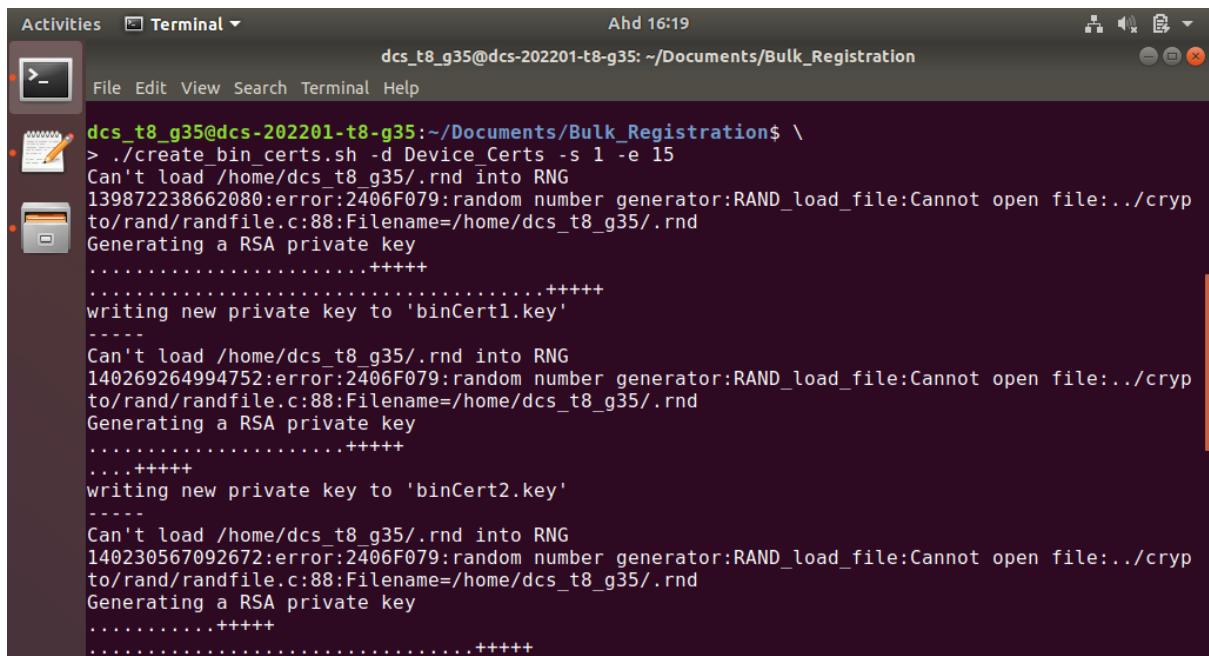
The screenshot shows a terminal window titled "Terminal" with the command history displayed. The terminal window is part of the Unity desktop environment, indicated by the "Activities" button and the docked application icons on the left. The terminal output is as follows:

```
Activities Terminal ▾ Jum 13:34  
root@danhish-garbage-bin-1: /home/danhish-garbage-bin-1/Documents/client_device  
danish-garbage-bin-1@danhish-garbage-bin-1:~$ sudo su  
[sudo] password for danish-garbage-bin-1:  
root@danhish-garbage-bin-1:/home/danhish-garbage-bin-1# source \  
> iot-bin-py-env/bin/activate  
(iot-bin-py-env) root@danhish-garbage-bin-1:/home/danhish-garbage-bin-1# cd Documents/client_ dev  
ice/  
(iot-bin-py-env) root@danhish-garbage-bin-1:/home/danhish-garbage-bin-1/Documents/client_device#
```

### 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.

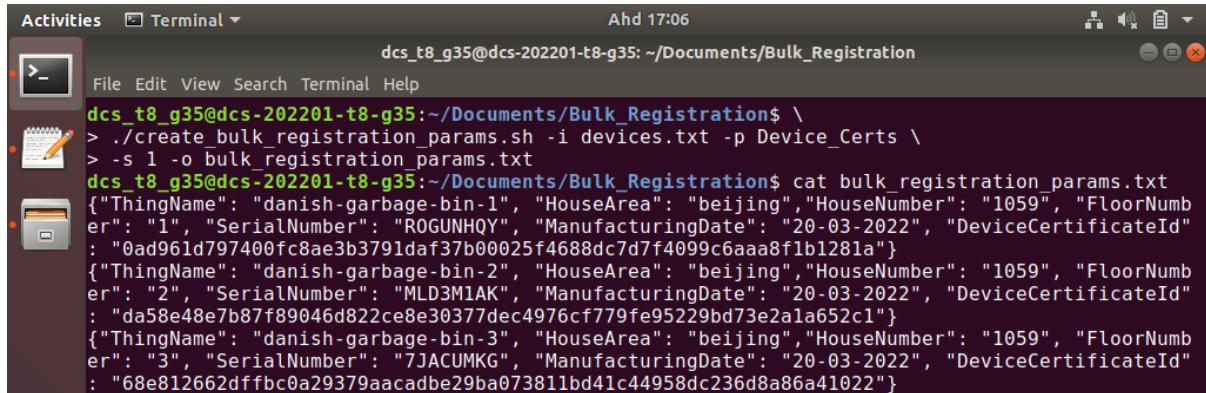
```
./create_bin_certs.sh -d Device_Certs -s 1 -e 15
```



```
Ahd 16:19
dcs_t8_g35@dcs-202201-t8-g35: ~/Documents/Bulk_Registration
File Edit View Search Terminal Help
dcs_t8_g35@dcs-202201-t8-g35:~/Documents/Bulk_Registration$ 
> ./create_bin_certs.sh -d Device_Certs -s 1 -e 15
Can't load /home/dcs_t8_g35/.rnd into RNG
139872238662080:error:2406F079:random number generator:RAND_load_file:Cannot open file:../crypto/rand/randfile.c:88:Filename=/home/dcs_t8_g35/.rnd
Generating a RSA private key
+++++
writing new private key to 'binCert1.key'
-----
Can't load /home/dcs_t8_g35/.rnd into RNG
140269264994752:error:2406F079:random number generator:RAND_load_file:Cannot open file:../crypto/rand/randfile.c:88:Filename=/home/dcs_t8_g35/.rnd
Generating a RSA private key
+++++
writing new private key to 'binCert2.key'
-----
Can't load /home/dcs_t8_g35/.rnd into RNG
140230567092672:error:2406F079:random number generator:RAND_load_file:Cannot open file:../crypto/rand/randfile.c:88:Filename=/home/dcs_t8_g35/.rnd
Generating a RSA private key
+++++
```

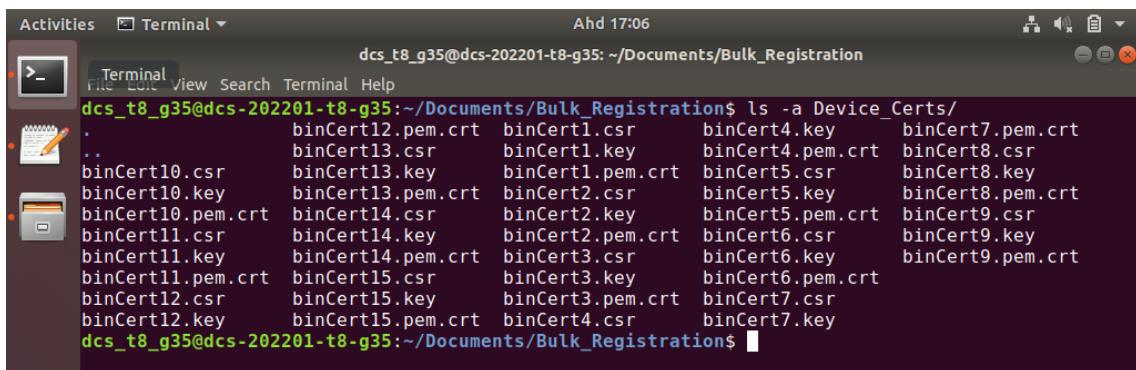
- 3) Execute the command below to generate the bulk registration parameter values required for the bulk registration. The **i** argument indicates the path to the file containing the device metadata; the **p** argument indicates the directory where the CSR are stored; while the **s** argument indicates 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_registration_params.sh -i devices.txt -p Device_Certs -s 1 -o
bulk_registration_params.txt
```



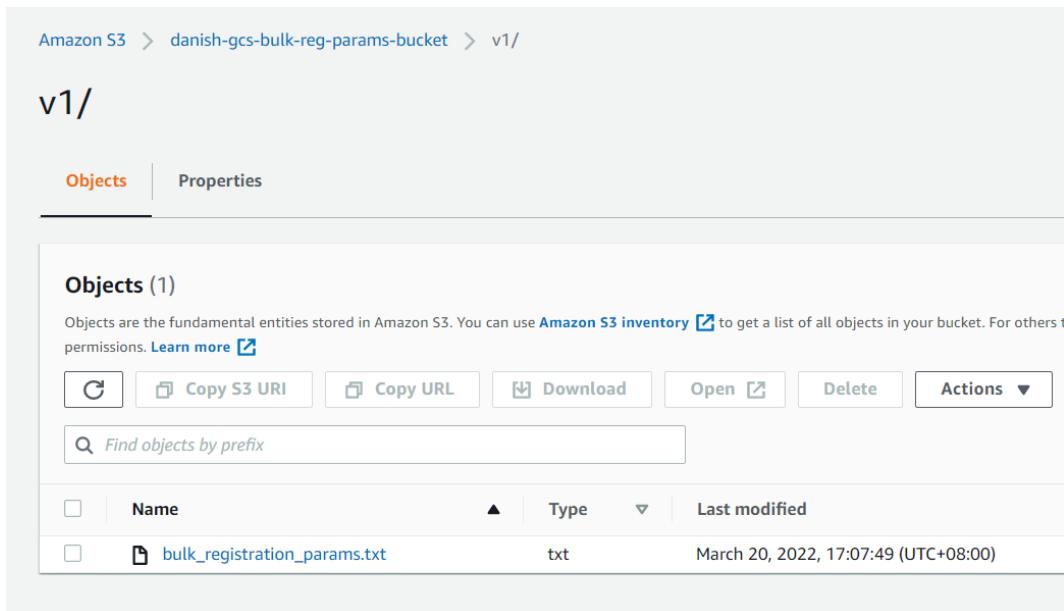
```
Ahd 17:06
dcs_t8_g35@dcs-202201-t8-g35: ~/Documents/Bulk_Registration
dcs_t8_g35@dcs-202201-t8-g35:~/Documents/Bulk_Registration$ ./create_bulk_registration_params.sh -i devices.txt -p Device_Certs \
> -s 1 -o bulk_registration_params.txt
dcs_t8_g35@dcs-202201-t8-g35:~/Documents/Bulk_Registration$ cat bulk_registration_params.txt
[{"ThingName": "danish-garbage-bin-1", "HouseArea": "beijing", "HouseNumber": "1059", "FloorNumber": "1", "SerialNumber": "ROGUNHQY", "ManufacturingDate": "20-03-2022", "DeviceCertificateId": "0ad961d797400fc8ae3b3791daf37b00025f4688dc7d7f4099c6aaa8f1b1281a"}, {"ThingName": "danish-garbage-bin-2", "HouseArea": "beijing", "HouseNumber": "1059", "FloorNumber": "2", "SerialNumber": "MLD3MIAK", "ManufacturingDate": "20-03-2022", "DeviceCertificateId": "da58e48e7b87f89046d822ce8e30377dec4976cff79fe95229bd73e2a1a652c1"}, {"ThingName": "danish-garbage-bin-3", "HouseArea": "beijing", "HouseNumber": "1059", "FloorNumber": "3", "SerialNumber": "7JACUMKG", "ManufacturingDate": "20-03-2022", "DeviceCertificateId": "68e812662dffbc0a29379aacadbe29ba073811bd41c44958dc236d8a86a41022"}]
```

- 4) The **Device\_Certs** directory now contains the private key, CSR, and the device certificate.



```
Ahd 17:06
dcs_t8_g35@dcs-202201-t8-g35: ~/Documents/Bulk_Registration
dcs_t8_g35@dcs-202201-t8-g35:~/Documents/Bulk_Registration$ ls -a Device_Certs/
.           binCert12.pem.crt  binCert1.csr      binCert4.key      binCert7.pem.crt
..          binCert13.csr      binCert1.key     binCert4.pem.crt  binCert8.csr
binCert10.csr          binCert13.key    binCert1.pem.crt  binCert5.csr      binCert8.key
binCert10.key          binCert13.pem.crt binCert2.csr      binCert5.key     binCert8.pem.crt
binCert10.pem.crt      binCert14.csr      binCert2.key     binCert5.pem.crt binCert9.csr
binCert11.csr          binCert14.key      binCert2.pem.crt binCert6.csr      binCert9.key
binCert11.key          binCert14.pem.crt binCert3.csr      binCert6.key     binCert9.pem.crt
binCert11.pem.crt      binCert15.csr      binCert3.key     binCert6.pem.crt binCert7.csr
binCert12.csr          binCert15.key      binCert3.pem.crt binCert7.csr
binCert12.key          binCert15.pem.crt binCert4.csr      binCert7.key
dcs_t8_g35@dcs-202201-t8-g35:~/Documents/Bulk_Registration$
```

- 5) Upload the file **bulk\_registration\_params.txt** to the S3 bucket.



Amazon S3 > danish-gcs-bulk-reg-params-bucket > v1/

v1/

**Objects** (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified
<input type="checkbox"/>	<a href="#">bulk_registration_params.txt</a>	txt	March 20, 2022, 17:07:49 (UTC+08:00)

- 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
```

```
Activities Terminal Ahd 17:09
dcs_t8_g35@dcs-202201-t8-g35: ~/Documents/Bulk_Registration
File Edit View Search Terminal Help
dcs_t8_g35@dcs-202201-t8-g35: ~/Documents/Bulk_Registration$ \
> aws iot start-thing-registration-task \
> --aws-iot-things-certificate file://bulk_reg_provisioning_template.json \
> --input-file-bucket "danish-gcs-bulk-reg-params-bucket" \
> --input-file-key "v1/bulk_registration_params.txt" \
> --role-arm arn:aws:iam::385235753061:role/DanishGCSBulkRegistrationAccessS3Role
{
    "taskId": "155f1ca7-bacd-42da-9f07-45f14ddbe9bc"
}
```

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.

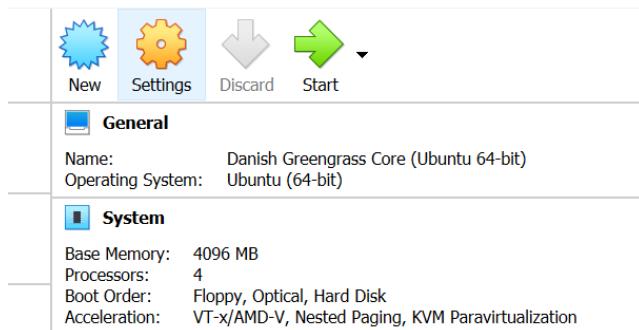
```
aws iot list-thing-registration-task-reports \
--task-id 155f1ca7-bacd-42da-9f07-45f14ddbe9bc \
--report-type RESULTS
aws iot list-thing-registration-task-reports \
--task-id 155f1ca7-bacd-42da-9f07-45f14ddbe9bc \
--report-type ERRORS
```

```
dcs_t8_g35@dcs-202201-t8-g35:~/Documents/Bulk_Registration$ \
> aws iot list-thing-registration-task-reports \
> --task-id 155f1ca7-bacd-42da-9f07-45f14ddbe9bc \
> --report-type RESULTS
{
    "resourceLinks": [
        "https://aws-iot-btp-prod-us-east-1.s3.us-east-1.amazonaws.com/385235753061/155f1ca7-b
acd-42da-9f07-45f14ddbe9bc/successful/155f1ca7-bacd-42da-9f07-45f14ddbe9bc-1?X-Amz-Security-To
ken=IQoJb3JpZ2luX2VjEHIAxCVzLWVhc3QtMSJGMEQCIDOKsB%2F0uRVKFLi89FQ6k98spCBhsFqtmsNDcHjv%2F5C7Ai
A9QNQS0Bvx9e6q90q012J3s4jBUpRxAjxPuYA%2FnQVNyirbAgjq%2F%2F%2F%2F%2F%2F%2F%2F%2F8BEAMaDD12ND
I4Nzc50TY2MSIM4nLHKf%2FeN9X7xbXKq8Ci6B5hYc3Vc2GcpTvVVUgl7Du%2FjxLtsiMu7qAcacFTF6WTQE8YfK7tql
9LdCaEne9rhyn92uOnFFxK0i8ps%2FQXAHNxDMYavW9dEQVqklIUudIiHYJEr7SaIF0WhdaAFMEZnDiMKlCuFcdoxPR
yzNCKCixxzwL6Cj4s74a0j7a42nhRCLR143inhfer1Ws7qLp0llbc%2B09soak2LPf7xLcQNaHDVaar41mwDdz7uGlzGdK
SCaG0bul0mQLPJVPgkPx%2FYn1JTnPnW%2BKtdBN8K%2B1NF8i4slclla2peQxtMpaIVXoiTKtqNakiOoiEGyiz9RioZgt
0$xaifIRFYGX7bLXI2CmnT%2FCK3VYiZuMd35vRLabaiNtEnVAX0yMT6u0oskFdhV0N3375HLjgEyZMPvf25EG0sABv4e5
DDStGCvSlQJSMs79sP8Jt0fVB9C38CysNnu6d6B00gGpypSP%2BPjVsHmYMY4MbcsZKfji%2BGLK1ghhvCRz%2
Fg3asqZZH9Th793in7tlgs9dN6oXeaZON0ieu6KML0C1cGHRiVbClmQraKCu2tSLq4r0wg1aL43Kw9tBNJB6cTa8X9%2F
9UlCEFZQH%2FpmXLjooTGCbEtiphIeq%2FdKrrC7eqkEH6ax15x%2FCyMtfvhabLV1l7%2B07IUkE%2BeU&X-Amz-Alg
orithm=AWS4-HMAC-SHA256&X-Amz-Date=20220320T091227Z&X-Amz-SignedHeaders=host&X-Amz-Expires=864
00&X-Amz-Credential=ASIAT3CGHZFWRM6BDH2C%2F20220320%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Sign
ature=cdd9a9849cb0b5d78bf40d38f358342c83e816e3bfa759de536271d6c69b6251"
    ],
    "reportType": "RESULTS"
}
dcs_t8_g35@dcs-202201-t8-g35:~/Documents/Bulk_Registration$ \
> aws iot list-thing-registration-task-reports \
> --task-id 155f1ca7-bacd-42da-9f07-45f14ddbe9bc \
> --report-type ERRORS
{
    "resourceLinks": [],
    "reportType": "ERRORS"
}
```

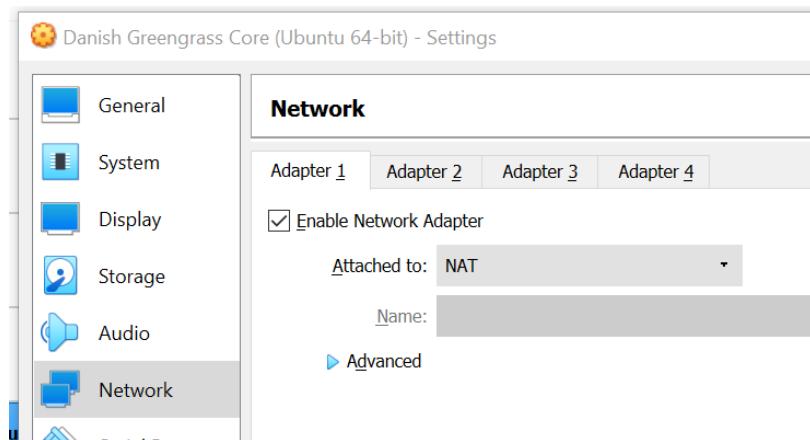
### 3.2 Establish local connection between client devices and core devices

To simulate LAN environment between two VirtualBox virtual machines,

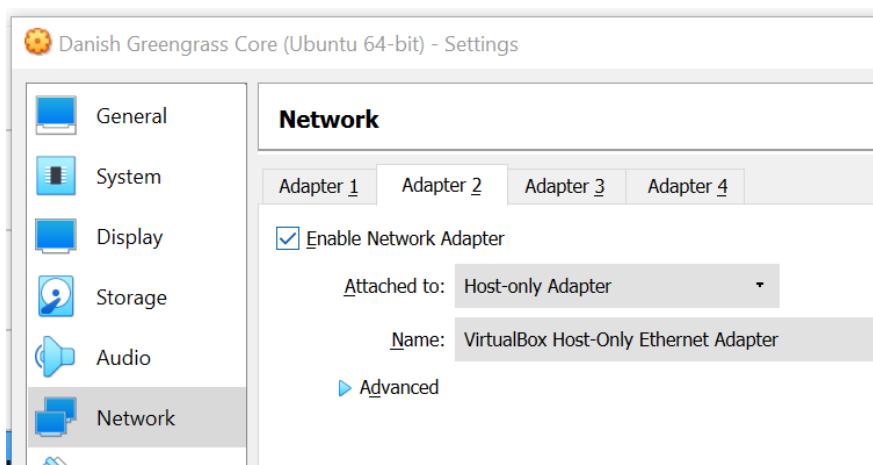
- 1) Open VirtualBox. Access the settings of the VirtualBox and select **Network**.



- 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.



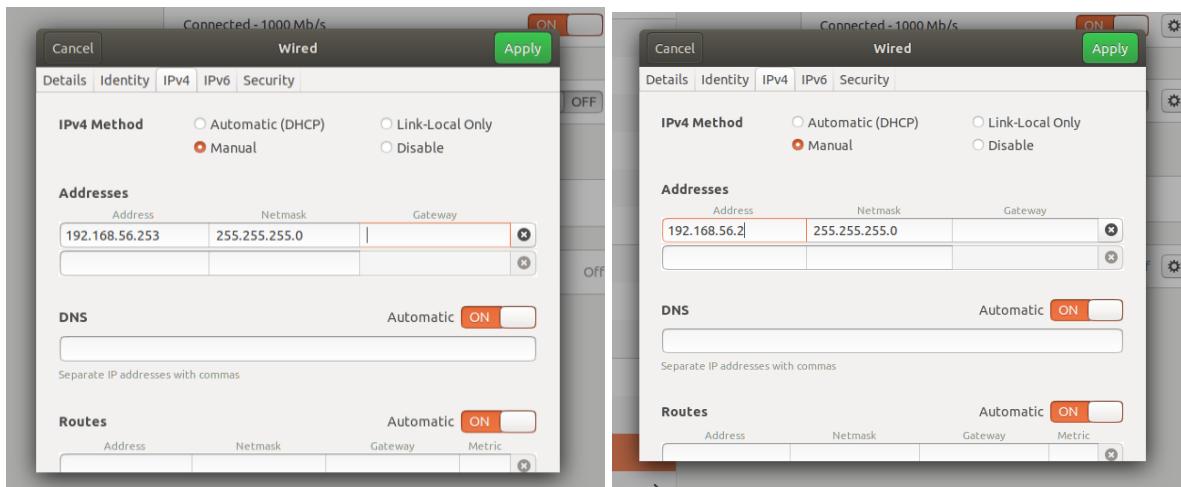
- 3) **Enable** and attach the **Adapter 2** to **Host-only Adapter** to allow communication between virtual machines as if they were in the same LAN (Oracle Corporation, 2022).



- 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:cff:fe199%20(Preferred)
IPv4 Address. . . . . : 192.168.56.1(Preferred)
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :
DHCPv6 IAID . . . . . : 50987047
```

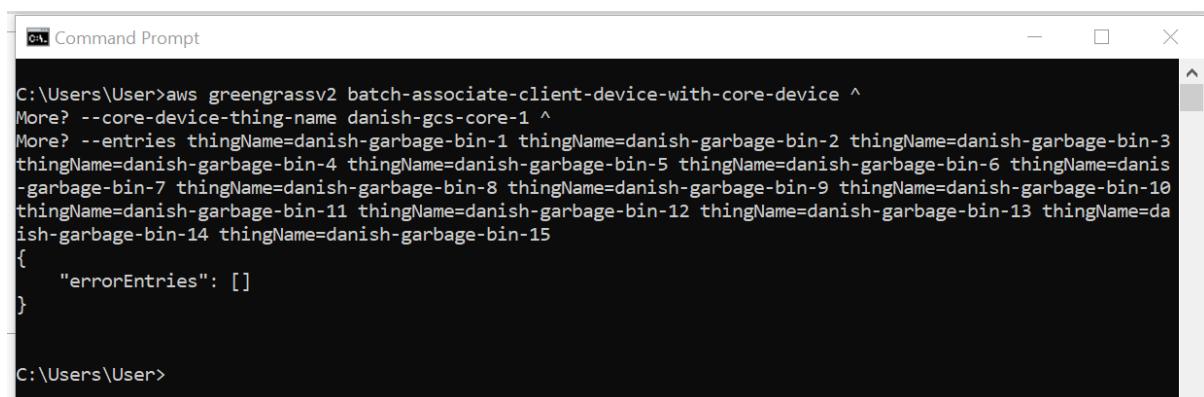
7) Ping each other to check the network configuration.

```
dcs_t8_g35@dcs-202201-t8-g35:~$ ping 192.168.56.2
PING 192.168.56.2 (192.168.56.2) 56(84) bytes of data.
64 bytes from 192.168.56.2: icmp_seq=1 ttl=64 time=3.03 ms
64 bytes from 192.168.56.2: icmp_seq=2 ttl=64 time=3.03 ms
64 bytes from 192.168.56.2: icmp_seq=3 ttl=64 time=1.71 ms
64 bytes from 192.168.56.2: icmp_seq=4 ttl=64 time=6.06 ms
^C
--- 192.168.56.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3011ms
rtt min/avg/max/mdev = 1.719/3.463/6.065/1.595 ms
```

```
danish-garbage-bin-1@danish-garbage-bin-1:~$ ping 192.168.56.253
PING 192.168.56.253 (192.168.56.253) 56(84) bytes of data.
64 bytes from 192.168.56.253: icmp_seq=1 ttl=64 time=1.71 ms
64 bytes from 192.168.56.253: icmp_seq=2 ttl=64 time=3.43 ms
64 bytes from 192.168.56.253: icmp_seq=3 ttl=64 time=2.85 ms
64 bytes from 192.168.56.253: icmp_seq=4 ttl=64 time=4.53 ms
^C
--- 192.168.56.253 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3024ms
rtt min/avg/max/mdev = 1.712/3.132/4.531/1.018 ms
```

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
```



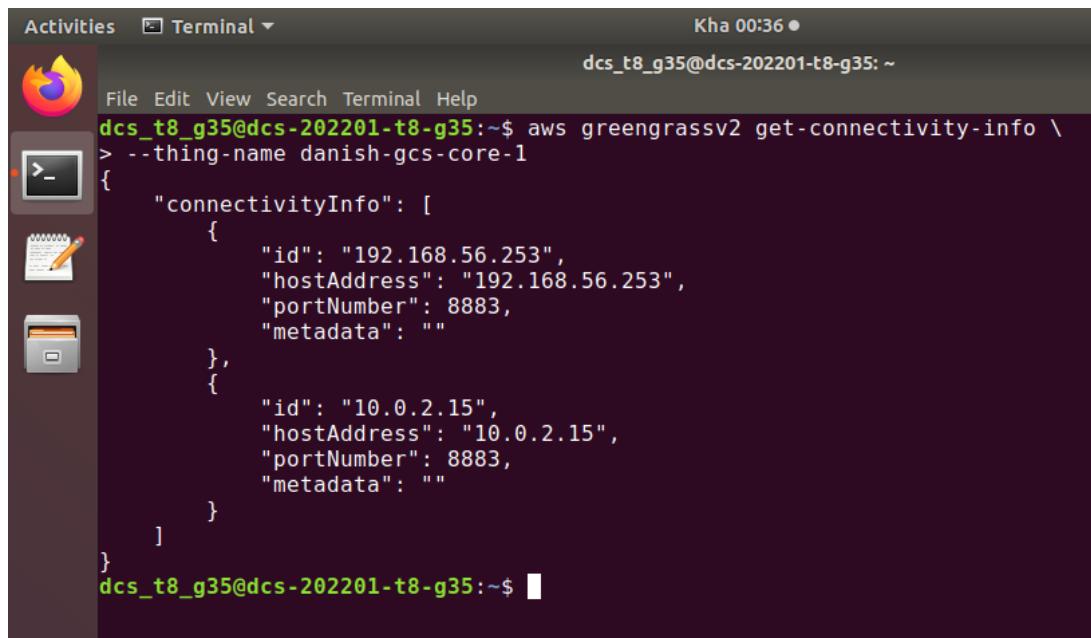
```
C:\Users\User>aws greengrassv2 batch-associate-client-device-with-core-device ^
More? --core-device-thing-name danish-gcs-core-1 ^
More? --entries thingName=danish-garbage-bin-1 thingName=danish-garbage-bin-2 thingName=danish-garbage-bin-3
thingName=danish-garbage-bin-4 thingName=danish-garbage-bin-5 thingName=danish-garbage-bin-6 thingName=danis-
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=da-
sh-garbage-bin-14 thingName=danish-garbage-bin-15
{
    "errorEntries": []
}

C:\Users\User>
```

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
```



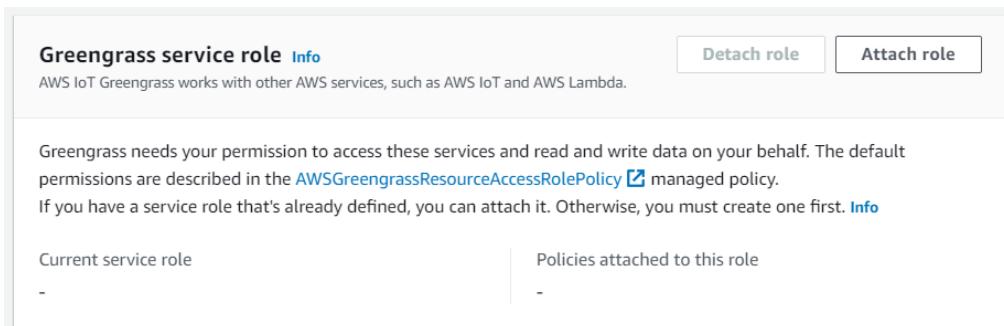
A screenshot of a Linux desktop environment showing a terminal window. The terminal window title is "Activities Terminal". The terminal session shows the command "aws greengrassv2 get-connectivity-info --thing-name danish-gcs-core-1" being run, followed by its JSON output. The output indicates two connectivity entries: one for the host address 192.168.56.253 (port 8883) and another for the local adapter address 10.0.2.15 (port 8883). The terminal prompt "dcs\_t8\_g35@dc..." is visible at the bottom.

```
dcs_t8_g35@dc...:~$ aws greengrassv2 get-connectivity-info \
> --thing-name danish-gcs-core-1
{
    "connectivityInfo": [
        {
            "id": "192.168.56.253",
            "hostAddress": "192.168.56.253",
            "portNumber": 8883,
            "metadata": ""
        },
        {
            "id": "10.0.2.15",
            "hostAddress": "10.0.2.15",
            "portNumber": 8883,
            "metadata": ""
        }
    ]
}
dcs_t8_g35@dc...:~$
```

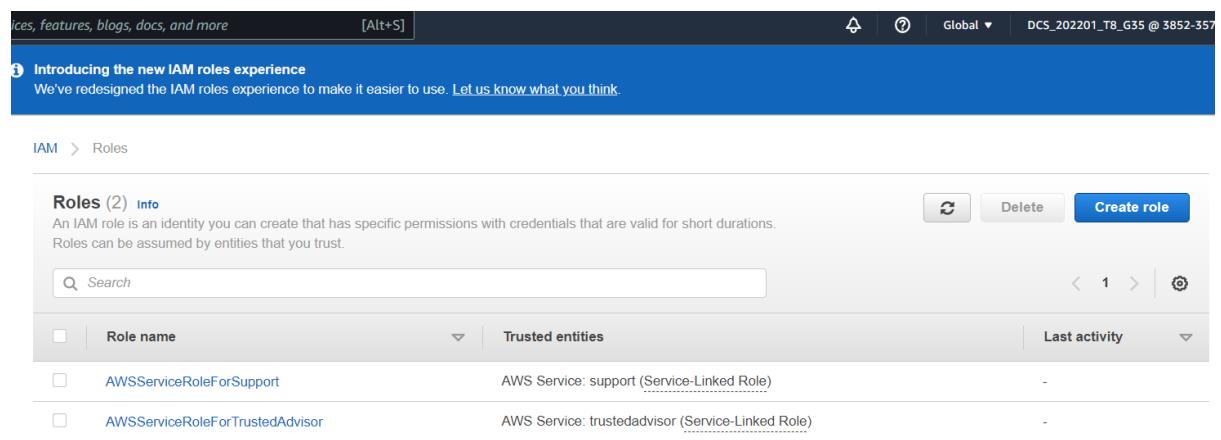
## Section 4: Greengrass Core Device Setup

### 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**.

## Select trusted entity

### Trusted entity type

<input checked="" type="radio"/> AWS service Allow AWS services like EC2, Lambda, or others to perform actions in this account.	<input type="radio"/> AWS account Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
--	---

<input type="radio"/> SAML 2.0 federation Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.	<input type="radio"/> Custom trust policy Create a custom trust policy to enable others to perform actions in this account.
---	--

### Use case

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

#### Common use cases

- EC2  
Allows EC2 instances to call AWS services on your behalf.
- Lambda  
Allows Lambda functions to call AWS services on your behalf.

#### Use cases for other AWS services:

- Greengrass  
Allows Greengrass to call AWS services on your behalf.
- Greengrass  
Allows Greengrass to call AWS services on your behalf.

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

## Add permissions

**Permissions policies** (Selected 1/733)  
Choose one or more policies to attach to your new role.

Filter policies by property or policy name and press enter

"AWSGreengrassResourceAccessRolePolicy" X Clear filters

Policy name	Type	Description
AWSGreengrassRe...	AWS m...	Policy for AWS Greengrass service role which allows access to rela

- 6) Name the role as **Greengrass\_ServiceRole**. Review the role details before click **Create role**.

## Name, review, and create

### Role details

Role name  
Enter a meaningful name to identify this role.

Greengrass\_ServiceRole

Maximum 128 characters. Use alphanumeric and '+,-,@-' characters.

- 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.

The screenshot shows the 'Greengrass service role' configuration page. At the top, there are 'Info' and 'AWS IoT Greengrass works with other AWS services, such as AWS IoT and AWS Lambda.' buttons, along with 'Detach role' and 'Change role' buttons. Below this, a note states: 'Greengrass needs your permission to access these services and read and write data on your behalf. The default permissions are described in the [AWSGreengrassResourceAccessRolePolicy](#) managed policy. If you have a service role that's already defined, you can attach it. Otherwise, you must create one first. [Info](#)'.

A green success message box contains the text: 'Greengrass service role has been updated successfully.' with a close button 'X'.

Below the message, two sections are shown: 'Current service role' (Greengrass\_ServiceRole) and 'Policies attached to this role' (AWSGreengrassResourceAccessRolePolicy).

## 4.2 Check Device Requirements

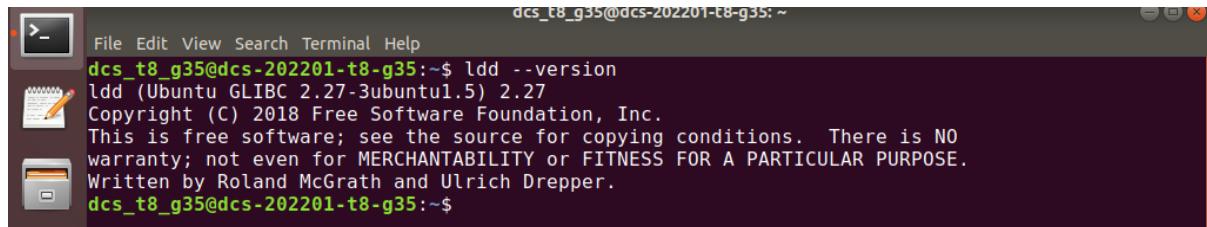
Before proceeding to installing the Greengrass Core Software in the edge device, ensure that the device meets the following requirements:

- 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
```

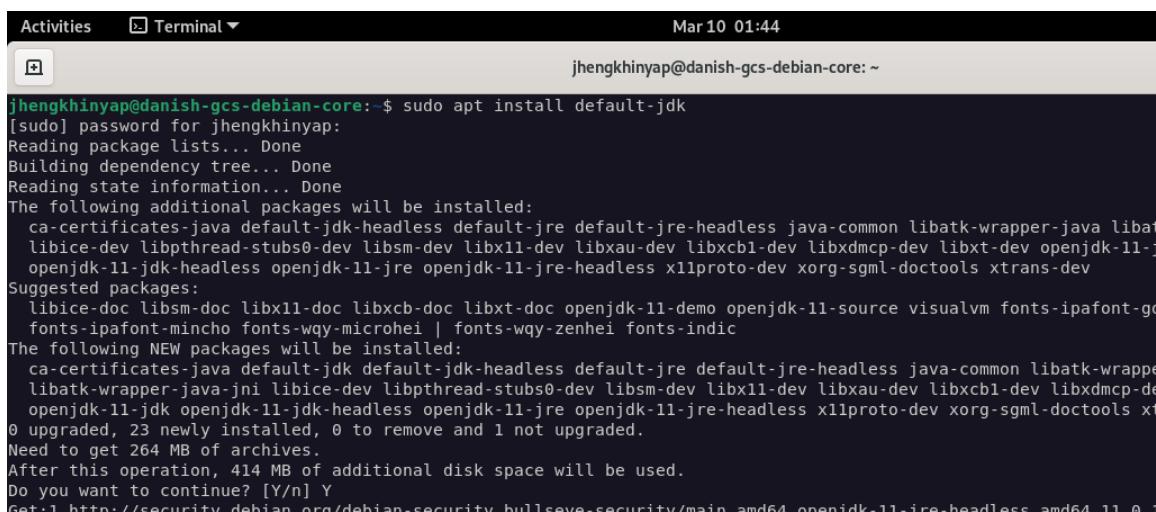


```
dcs_t8_g35@dcs-202201-t8-g35:~$ ldd --version
ldd (Ubuntu GLIBC 2.27-3ubuntu1.5) 2.27
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
dcs_t8_g35@dcs-202201-t8-g35:~$
```

#### 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.

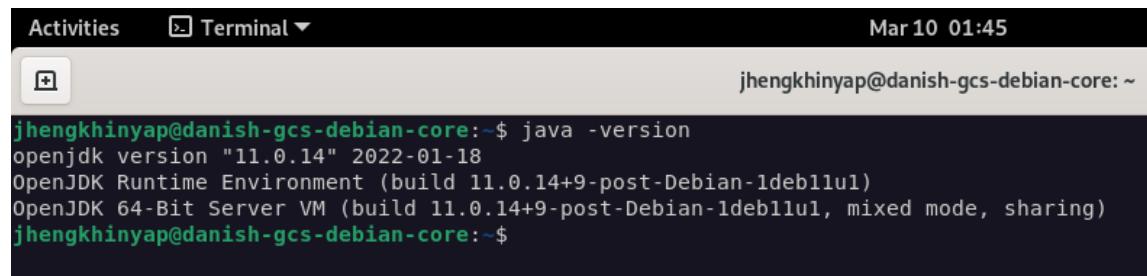
```
sudo apt install default-jdk
java -version
```



```
Activities Terminal Mar 10 01:44
jhengkhinyap@danish-gcs-debian-core:~
```

```
jhengkhinyap@danish-gcs-debian-core:~$ sudo apt install default-jdk
[sudo] password for jhengkhinyap:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java default-jdk-headless default-jre default-jre-headless java-common libatk-wrapper-java libatk-wrapper-java-jni libice-dev libpthread-stubs0-dev libsm-dev libx11-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-11-jdk-headless openjdk-11-jre openjdk-11-jre-headless x11proto-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  libice-doc libsm-doc libxcb-doc libxext-doc openjdk-11-demo openjdk-11-source visualvm fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei | fonts-wqy-zenhei fonts-indic
The following NEW packages will be installed:
  ca-certificates-java default-jdk default-jdk-headless default-jre default-jre-headless java-common libatk-wrapper-java libatk-wrapper-java-jni libice-dev libpthread-stubs0-dev libsm-dev libx11-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-11-jdk openjdk-11-jdk-headless openjdk-11-jre openjdk-11-jre-headless x11proto-dev xorg-sgml-doctools xt
0 upgraded, 23 newly installed, 0 to remove and 1 not upgraded.
Need to get 264 MB of archives.
After this operation, 414 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://security.debian.org/debian-security/bullseye-security/main amd64 openjdk-11-jre-headless amd64 11.0.14+9-post-Debian-1deb11u1 [10.3MB]
```

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



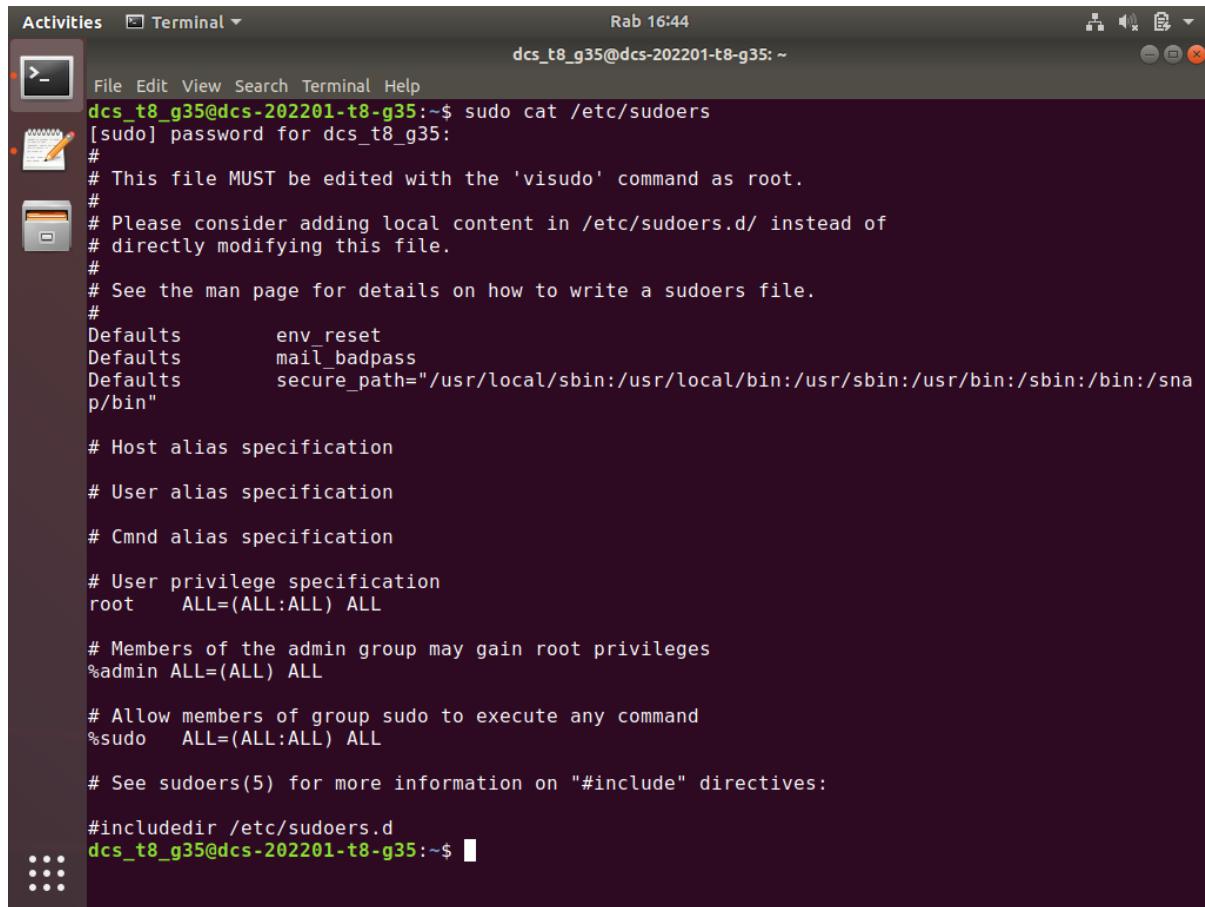
```
Activities Terminal Mar 10 01:45
jhengkhinyap@danish-gcs-debian-core:~
```

```
jhengkhinyap@danish-gcs-debian-core:~$ java -version
openjdk version "11.0.14" 2022-01-18
OpenJDK Runtime Environment (build 11.0.14+9-post-Debian-1deb11u1)
OpenJDK 64-Bit Server VM (build 11.0.14+9-post-Debian-1deb11u1, mixed mode, sharing)
jhengkhinyap@danish-gcs-debian-core:~$
```

#### 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
```



The screenshot shows a terminal window titled "Terminal" with the command "sudo cat /etc/sudoers" being run. The terminal output displays the contents of the /etc/sudoers file, which includes various configuration directives such as Defaults, Host alias specification, User alias specification, Cmnd alias specification, and User privilege specification. A password prompt "[sudo] password for dcs\_t8\_g35:" is visible, followed by the actual sudoers file content. The terminal window is part of a desktop environment with icons for Activities, Home, and Dash.

```
dcs_t8_g35@dcs-202201-t8-g35:~$ sudo cat /etc/sudoers
[sudo] password for dcs_t8_g35:
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults      env_reset
Defaults      mail_badpass
Defaults      secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root      ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin    ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo    ALL=(ALL:ALL) ALL

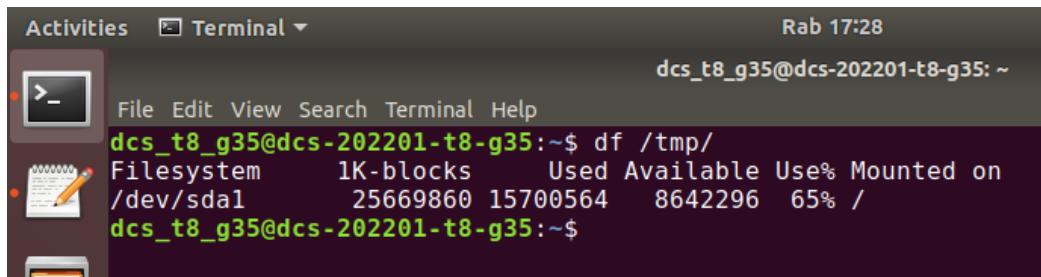
# See sudoers(5) for more information on "#include" directives:
#include /etc/sudoers.d
dcs_t8_g35@dcs-202201-t8-g35:~$
```

- 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.



```
Activities Terminal Rab 17:28
dcs_t8_g35@dcs-202201-t8-g35:~$ df /tmp/
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/sda1 25669860 15700564 8642296 65% /
dcs_t8_g35@dcs-202201-t8-g35:~$
```

- 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.



```
Activities Terminal Rab 17:20
dcs_t8_g35@dcs-202201-t8-g35:~$ cat /proc/mounts | grep tmpfs
tmpfs /dev tmpfs rw,nosuid,relatime,size=1987880k,nr_inodes=496970,mode=755 0 0
tmpfs /run tmpfs rw,nosuid,noexec,relatime,size=402568k,mode=755 0 0
tmpfs /dev/shm tmpfs rw,nosuid,nodev 0 0
tmpfs /run/lock tmpfs rw,nosuid,nodev,noexec,relatime,size=5120k 0 0
tmpfs /sys/fs/cgroup tmpfs ro,nosuid,nodev,noexec,mode=755 0 0
tmpfs /run/user/121 tmpfs rw,nosuid,nodev,relatime,size=402564k,mode=700,uid=121,gid=125 0 0
tmpfs /run/user/1000 tmpfs rw,nosuid,nodev,relatime,size=402564k,mode=700,uid=1000,gid=1000 0
dcs_t8_g35@dcs-202201-t8-g35:~$
```

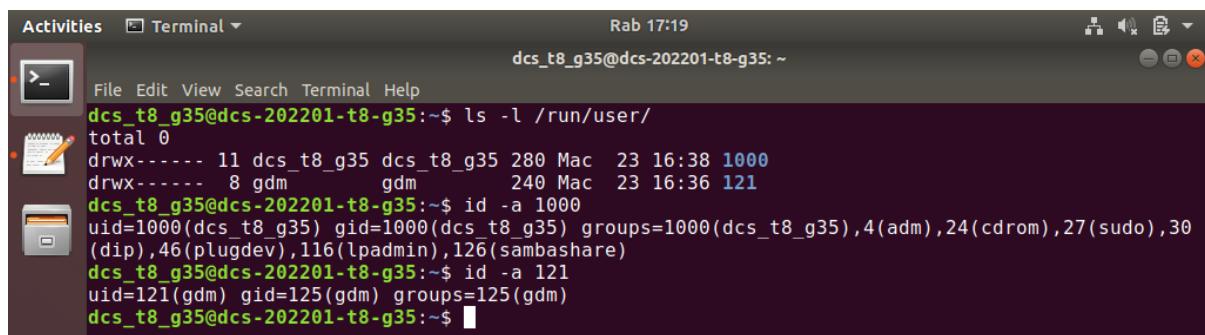
- 5) There is some prerequisite to understand the output above:

- 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
```



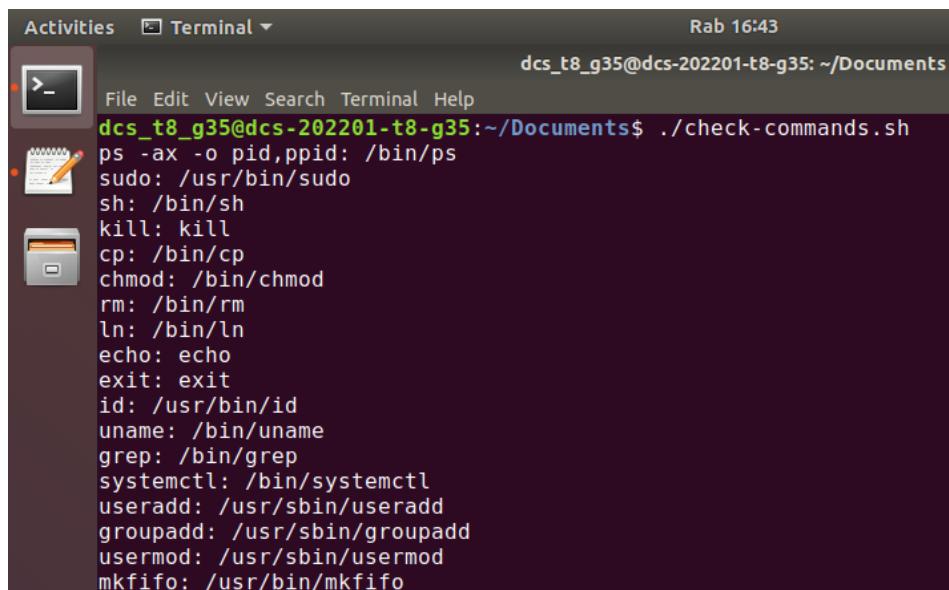
A screenshot of a Linux terminal window titled "Terminal". The window shows the command "ls -l /run/user/" followed by its output, which lists two directories: one for user 1000 and one for user 121. Below that, the command "id -a 1000" is run, showing the user's ID, group ID, and supplementary groups. Then "id -a 121" is run, showing the same information for user 121. The terminal window is part of a desktop environment with icons for Activities, Terminal, File, Edit, View, Search, Terminal, and Help.

```
dcs_t8_g35@dcs-202201-t8-g35:~$ ls -l /run/user/
total 0
drwx----- 11 dcs_t8_g35 dcs_t8_g35 280 Mac 23 16:38 1000
drwx----- 8 gdm      gdm      240 Mac 23 16:36 121
dcs_t8_g35@dcs-202201-t8-g35:~$ id -a 1000
uid=1000(dcs_t8_g35) gid=1000(dcs_t8_g35) groups=1000(dcs_t8_g35),4(adm),24(cdrom),27(sudo),30
(dip),46(plugdev),116(lpadmin),126(sambashare)
dcs_t8_g35@dcs-202201-t8-g35:~$ id -a 121
uid=121(gdm) gid=125(gdm) groups=125(gdm)
dcs_t8_g35@dcs-202201-t8-g35:~$
```

#### 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
```



A screenshot of a Linux terminal window titled "Terminal". The window shows the command "./check-commands.sh" followed by a list of various system commands that were found to be available. The commands listed include ps, sudo, sh, kill, cp, chmod, rm, ln, echo, exit, id, uname, grep, systemctl, useradd, groupadd, usermod, and mkfifo.

```
dcs_t8_g35@dcs-202201-t8-g35:~/Documents$ ./check-commands.sh
ps -ax -o pid,ppid: /bin/ps
sudo: /usr/bin/sudo
sh: /bin/sh
kill: kill
cp: /bin/cp
chmod: /bin/chmod
rm: /bin/rm
ln: /bin/ln
echo: echo
exit: exit
id: /usr/bin/id
uname: /bin/uname
grep: /bin/grep
systemctl: /bin/systemctl
useradd: /usr/sbin/useradd
groupadd: /usr/sbin/groupadd
usermod: /usr/sbin/usermod
mkfifo: /usr/bin/mkfifo
```

#### 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
```

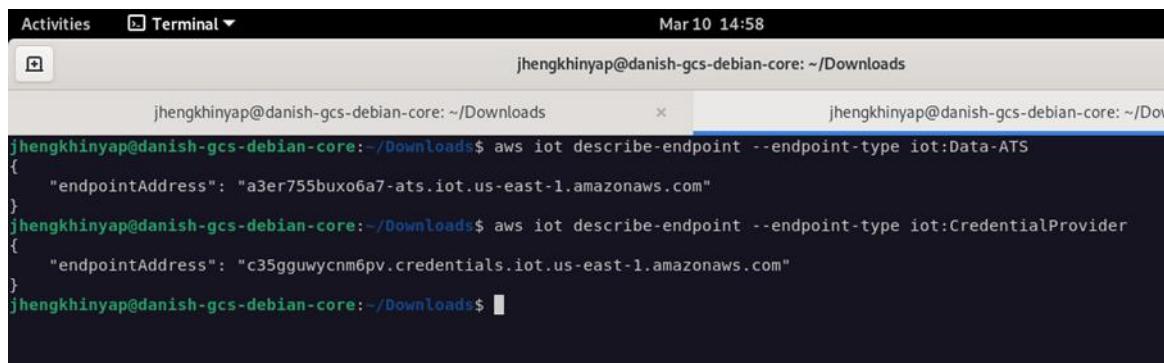
```
jhengkhinyap@danish-gcs-debian-core:~/Downloads$ curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-  
nucleus-latest.zip > greengrass-nucleus-latest.zip  
jhengkhinyap@danish-gcs-debian-core:~/Downloads$ unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm g  
reengrass-nucleus-latest.zip  
Archive: greengrass-nucleus-latest.zip  
  inflating: GreengrassInstaller/LICENSE  
  inflating: GreengrassInstaller/NOTICE  
  inflating: GreengrassInstaller/README.md  
  inflating: GreengrassInstaller/THIRD-PARTY-LICENSES  
  inflating: GreengrassInstaller/bin/greengrass.exe  
  inflating: GreengrassInstaller/bin/greengrass.service.template  
  inflating: GreengrassInstaller/bin/greengrass.xml.template  
  inflating: GreengrassInstaller/bin/loader  
  inflating: GreengrassInstaller/bin/loader.cmd  
  inflating: GreengrassInstaller/conf/recipe.yaml  
  inflating: GreengrassInstaller/lib/Greengrass.jar  
jhengkhinyap@danish-gcs-debian-core:~/Downloads$ java -jar ./GreengrassInstaller/lib/Greengrass.jar --version  
AWS Greengrass v2.5.3  
jhengkhinyap@danish-gcs-debian-core:~/Downloads$
```

- 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
```

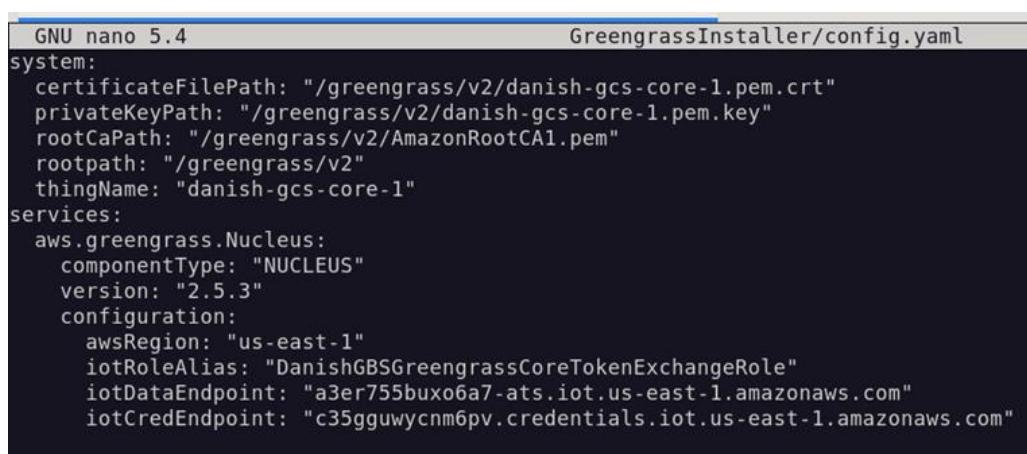


The screenshot shows a terminal window titled "Terminal" with the status bar indicating "Mar 10 14:58". The user is logged in as "jhengkhinyap@danish-gcs-debian-core: ~/Downloads". The terminal displays two commands run sequentially:

```
jhengkhinyap@danish-gcs-debian-core:~/Downloads$ aws iot describe-endpoint --endpoint-type iot:Data-ATS
{
  "endpointAddress": "a3er755buxo6a7-ats.iot.us-east-1.amazonaws.com"
}
jhengkhinyap@danish-gcs-debian-core:~/Downloads$ aws iot describe-endpoint --endpoint-type iot:CredentialProvider
{
  "endpointAddress": "c35gguywcnm6pv.credentials.iot.us-east-1.amazonaws.com"
}
```

- 6) Execute the command below to edit the **config.yaml** in the **GreengrassInstaller** directory. Replace the values of **iotDataEndpoint** 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
```

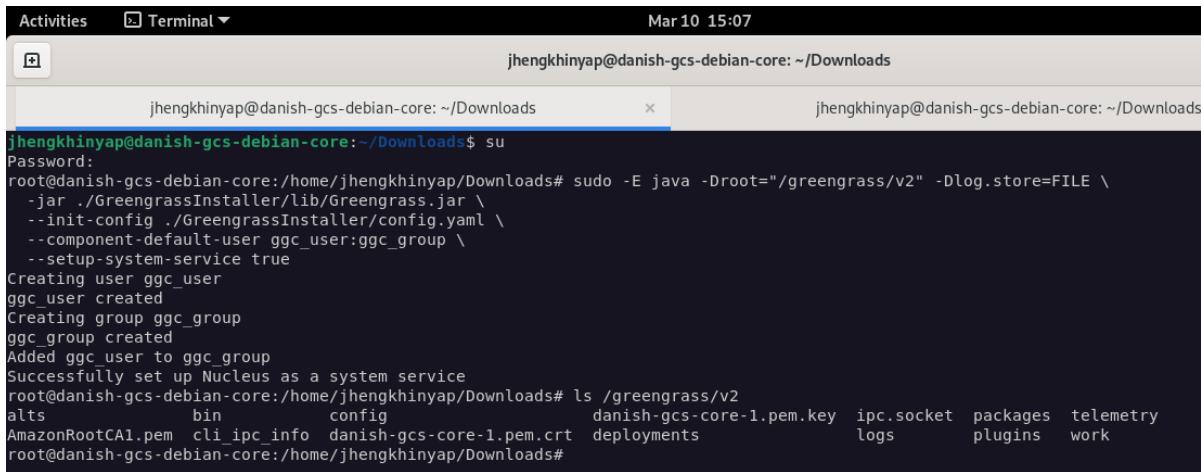


The screenshot shows a terminal window titled "GNU nano 5.4" with the file "GreengrassInstaller/config.yaml" open. The content of the file is a YAML configuration block:

```
system:
  certificateFilePath: "/greengrass/v2/danish-gcs-core-1.pem.crt"
  privateKeyPath: "/greengrass/v2/danish-gcs-core-1.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "danish-gcs-core-1"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.5.3"
    configuration:
      awsRegion: "us-east-1"
      iotRoleAlias: "DanishGBSGreengrassCoreTokenExchangeRole"
      iotDataEndpoint: "a3er755buxo6a7-ats.iot.us-east-1.amazonaws.com"
      iotCredEndpoint: "c35gguywcnm6pv.credentials.iot.us-east-1.amazonaws.com"
```

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 **gcc\_group** with a user named **gcc\_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 gcc_user:gcc_group \
--deploy-dev-tools true \
--setup-system-service true
```



The screenshot shows a terminal window titled "Terminal" with the date "Mar 10 15:07". The session is for user "jhengkhinyap" at host "danish-gcs-debian-core" with the command line "Downloads". The terminal displays the execution of the Java command to install Greengrass. It shows the creation of a user "gcc\_user", a group "gcc\_group", and adding "gcc\_user" to "gcc\_group". It also indicates that Nucleus was successfully set up as a system service. Finally, it lists the contents of the "/greengrass/v2" directory, which includes files like "alts", "bin", "config", "danhish-gcs-core-1.pem.key", "ipc.socket", "packages", "telemetry", "AmazonRootCA1.pem", "cli", "ipc\_info", "danhish-gcs-core-1.pem.crt", "deployments", "logs", "plugins", and "work".

```
jhengkhinyap@danish-gcs-debian-core: ~/Downloads
jhengkhinyap@danish-gcs-debian-core: ~/Downloads$ su
Password:
root@danish-gcs-debian-core:/home/jhengkhinyap/Downloads# sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
-jar ./GreengrassInstaller/lib/Greengrass.jar \
--init-config ./GreengrassInstaller/config.yaml \
--component-default-user gcc_user:gcc_group \
--setup-system-service true
Creating user gcc_user
gcc_user created
Creating group gcc_group
gcc_group created
Added gcc_user to gcc_group
Successfully set up Nucleus as a system service
root@danish-gcs-debian-core:/home/jhengkhinyap/Downloads# ls /greengrass/v2
alts      bin      config          danhish-gcs-core-1.pem.key  ipc.socket  packages  telemetry
AmazonRootCA1.pem  cli  ipc_info  danhish-gcs-core-1.pem.crt  deployments   logs      plugins  work
root@danish-gcs-debian-core:/home/jhengkhinyap/Downloads#
```

#### 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:

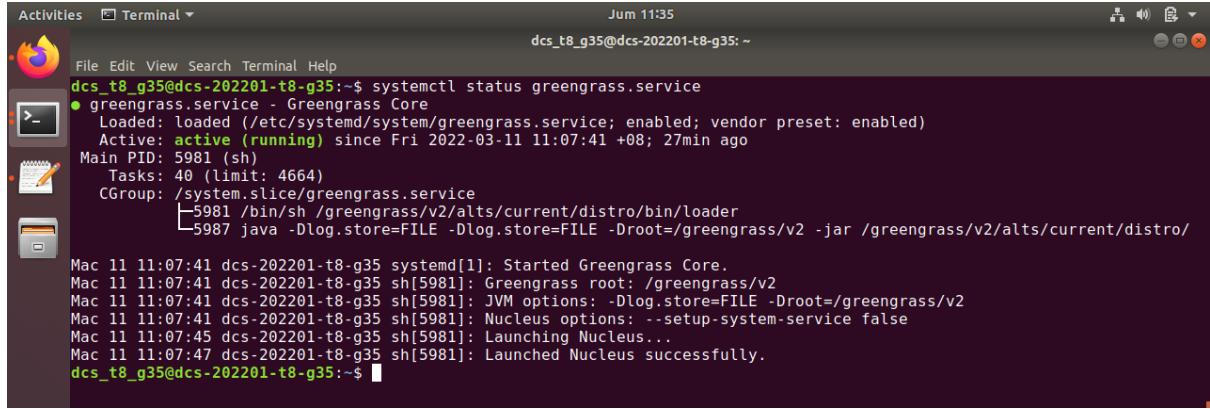
- If you specify `--provision`, the installer prints `Successfully configured Nucleus with provisioned resource details` if it configured the resources successfully.
- If you specify `--deploy-dev-tools`, the installer prints `Configured Nucleus to deploy aws.greengrass.Cli component` if it created the 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

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

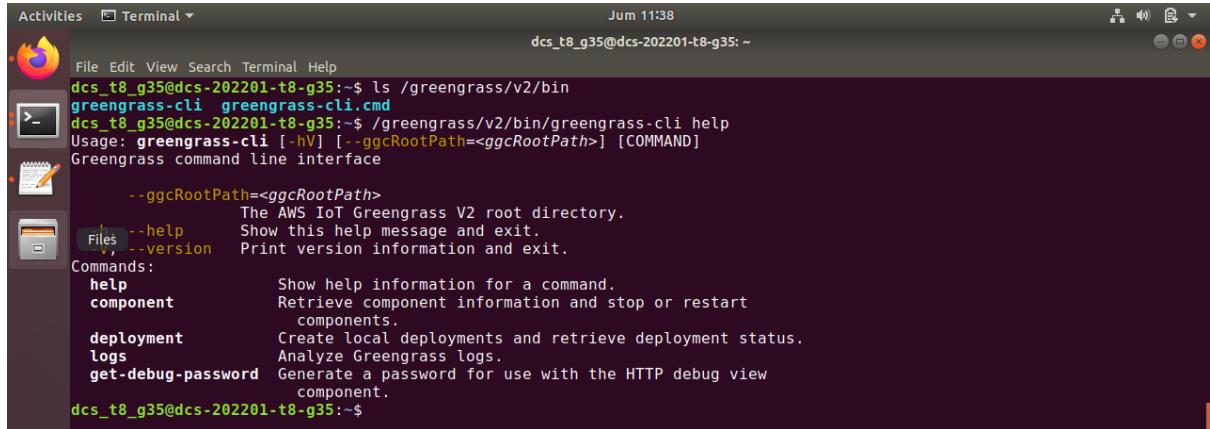
```
systemctl status greengrass.service
```



```
Jun 11:35
dcs_t8_g35@dcs-202201-t8-g35: ~
File Edit View Search Terminal Help
● greengrass.service - Greengrass Core
  Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor preset: enabled)
  Active: active (running) since Fri 2022-03-11 11:07:41 +08; 27min ago
    Main PID: 5981 (sh)
      Tasks: 40 (limit: 4664)
     CGroup: /system.slice/greengrass.service
             └─5981 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
                   ├─5987 java -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengrass/v2/alts/current/distro/
Mac 11 11:07:41 dcs-202201-t8-g35 systemd[1]: Started Greengrass Core.
Mac 11 11:07:41 dcs-202201-t8-g35 sh[5981]: Greengrass root: /greengrass/v2
Mac 11 11:07:41 dcs-202201-t8-g35 sh[5981]: JVM options: -Dlog.store=FILE -Droot=/greengrass/v2
Mac 11 11:07:41 dcs-202201-t8-g35 sh[5981]: Nucleus options: --setup-system-service false
Mac 11 11:07:45 dcs-202201-t8-g35 sh[5981]: Launching Nucleus...
Mac 11 11:07:47 dcs-202201-t8-g35 sh[5981]: Launched Nucleus successfully.
dcs_t8_g35@dcs-202201-t8-g35: ~
```

- 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
```



```
Jun 11:38
dcs_t8_g35@dcs-202201-t8-g35: ~
File Edit View Search Terminal Help
greengrass-cli greengrass-cli.cmd
dcs_t8_g35@dcs-202201-t8-g35: ~$ /greengrass/v2/bin/greengrass-cli help
Usage: greengrass-cli [-hV] [--ggcRootPath=<ggcRootPath>] [COMMAND]
Greengrass command line interface

--ggcRootPath=<ggcRootPath>
  The AWS IoT Greengrass V2 root directory.

  --help      Show this help message and exit.
  --version   Print version information and exit.

Commands:
  help        Show help information for a command.
  component   Retrieve component information and stop or restart
              components.
  deployment  Create local deployments and retrieve deployment status.
  logs        Analyze Greengrass logs.
  get-debug-password Generate a password for use with the HTTP debug view
                      component.
dcs_t8_g35@dcs-202201-t8-g35: ~
```

## 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.

Additional checksums

Off  
Amazon S3 will use a combination of MD5 checksums and Etags to verify data integrity.

On  
Specify a checksum function for additional data integrity validation.

Checksum function

Choose the checksum function that should be used to calculate the checksum value.

SHA-256 ▾

Precalculated value - *optional*

When you provide a precalculated value for a single object less than 16 MB, S3 compares it with the value it calculates using the selected checksum function. If the values don't match, the upload will not start. [Learn more](#) ↗

Enter value

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.DLRIInstallation
  - ii. danish.GCS.TrashClassification.ModelStore
  - iii. danish.GCS.TelemetryIngestion
  - iv. danish.GCS.TrashClassification

**Component information**  
Create a component from a recipe or import an AWS Lambda function. The component recipe is a YAML or JSON file that defines the component's details, dependencies, compatibility, and lifecycle. [Learn more](#)

**Component source**

Enter recipe as JSON  
Start with an example or enter your recipe.

Enter recipe as YAML  
Start with an example or enter your recipe.

Import Lambda function  
Import an AWS Lambda function as a component.

**Recipe**  
Your component artifacts must be available in an [S3 bucket](#), so you can link to them in the recipe. To deploy this component to a core device, that core device's role must allow access to the bucket. [Learn more](#)

```
1 "RecipeFormatVersion": "2020-01-25",
2   "ComponentName": "danish.GCS.DLRIInstallation",
3   "ComponentVersion": "1.0.0",
4   "ComponentType": "aws.greengrass.generic",
5   "ComponentDescription": "Run a script that installs Deep Learning Runtime (DLR) and its dependencies in a virtual environment on the core device.",
6   "ComponentPublisher": "Yap Jheng Khin",
```

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

**Deployment target**  
You can deploy to a single Greengrass core device or a group of core devices.

**Target type**

Core device

Thing group

**Target name**

danhish-gcs-core-1

[View core devices](#) to find a target.

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**.

## Select components

Select the components to deploy. The deployment includes the dependencies for each component that you select. You can edit the version and parameters of selected components in the next step.

**My components (4/4)**

<input checked="" type="checkbox"/>	Name
<input checked="" type="checkbox"/>	danish.GCS.TelemetryIngestion
<input checked="" type="checkbox"/>	danish.GCS.DLRIInstallation
<input checked="" type="checkbox"/>	danish.GCS.TrashClassification
<input checked="" type="checkbox"/>	danish.GCS.TrashClassification.ModelStore

## Public components (6/44)

<input checked="" type="checkbox"/>	Name
<input checked="" type="checkbox"/>	aws.greengrass.Cli
<input checked="" type="checkbox"/>	aws.greengrass.StreamManager
<input checked="" type="checkbox"/>	aws.greengrass.clientdevices.Auth
<input checked="" type="checkbox"/>	aws.greengrass.clientdevices.IPDetector
<input checked="" type="checkbox"/>	aws.greengrass.clientdevices.mqtt.Bridge
<input checked="" type="checkbox"/>	aws.greengrass.clientdevices.mqtt.Moquette

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

## Configure components - optional

You can configure the version and configuration parameters of each component to deploy. Components define default configuration parameters that you can customize in this deployment.

Selected components (10)				<a href="#">Configure component</a>
<input type="text"/> Find by name				< 1 >
Name	Version	Modified?		
aws.greengrass.Cli	2.5.3	-		
aws.greengrass.StreamManager	2.0.14	-		
aws.greengrass.clientdevices.Auth	2.0.4	-		
aws.greengrass.clientdevices.IPDetector	2.1.1	-		

- 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](#).)

### Configure aws.greengrass.clientdevices.Auth

Component version

Version: 2.0.4

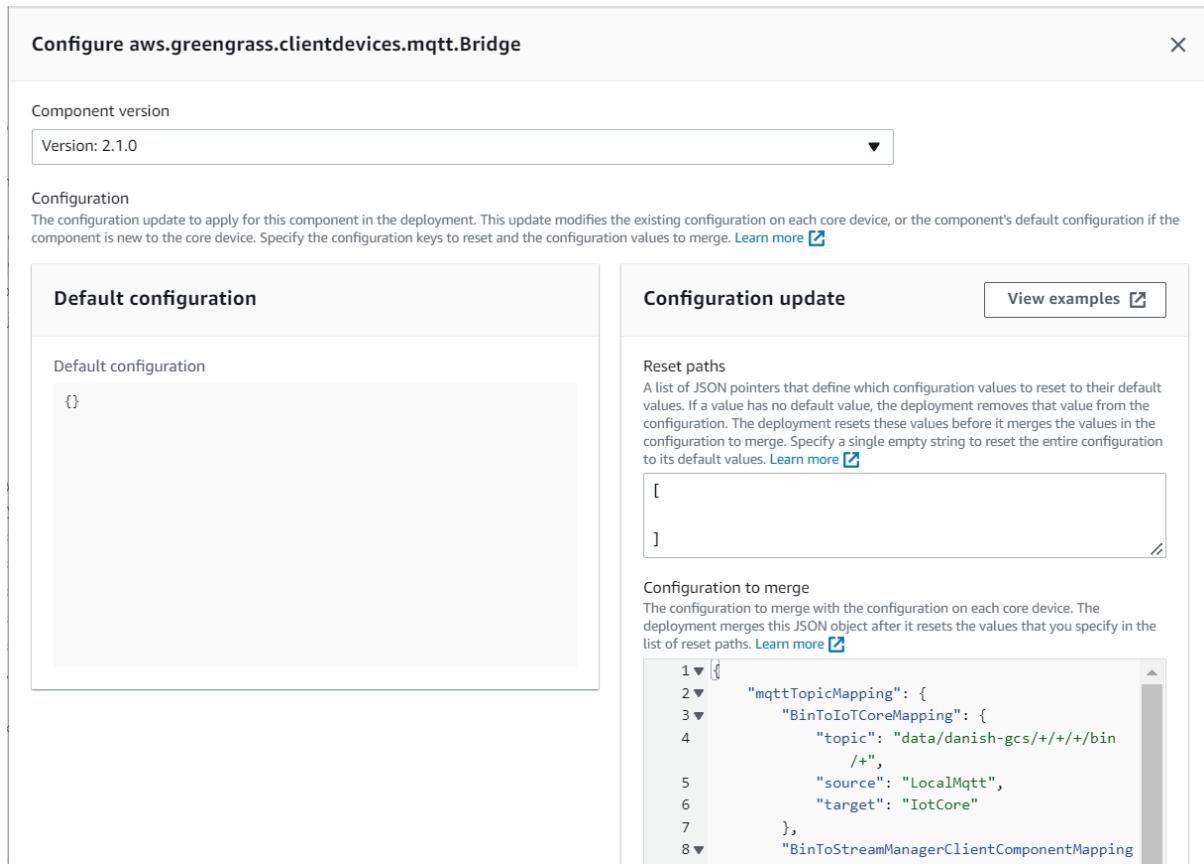
Configuration

The configuration update to apply for this component in the deployment. This update modifies the existing configuration on each core device, or the component's default configuration if the component is new to the core device. Specify the configuration keys to reset and the configuration values to merge. [Learn more](#)

<b>Default configuration</b>	<b>Configuration update</b>
Default configuration	Reset paths
{}	A list of JSON pointers that define which configuration values to reset to their default values. If a value has no default value, the deployment removes that value from the configuration. The deployment resets these values before it merges the values in the configuration to merge. Specify a single empty string to reset the entire configuration to its default values. <a href="#">Learn more</a>
	[ ] /
	Configuration to merge
	The configuration to merge with the configuration on each core device. The deployment merges this JSON object after it resets the values that you specify in the list of reset paths. <a href="#">Learn more</a>
	1 [ 2 "deviceGroups": { 3 "formatVersion": "2021-03-05", 4 "definitions": { } ]

- 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**.

AWS IoT > Greengrass > Core devices > danish-gcs-core-1

## danhish-gcs-core-1

**Overview**

Greengrass core devices are AWS IoT things that run the Greengrass Core software.

Thing danish-gcs-core-1	Status <span style="color: green;">Healthy</span>	Status reported 7 hours ago
Greengrass Core software version 2.5.3	Platform linux/amd64	

Components | **Deployments** | Thing groups | Client devices | Tags

**Deployments (1)**

Deployments define the software that run on each core device or group of core devices. These deployments target this Greengrass core device.

Deployment	Target	Status on this device	Status reported
Deployment for danish-gcs-core	danish-gcs-core	<span style="color: green;">Completed</span>	7 hours ago

- 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.

Components | Deployments | Thing groups | Client devices | Tags

**Components (11)**

This Greengrass core device runs these components. To edit the components on this core device, create a deployment to it or to one of its thing groups.

Name	Version	Status
danish.GCS.TrashClassification	1.0.1	<span style="color: red;">✖ Broken</span>
aws.greengrass.clientdevices.IPPDetector	2.1.1	<span style="color: green;">⌚ Running</span>
aws.greengrass.Cli	2.5.3	<span style="color: green;">⌚ Running</span>
danish.GCS.TelemetryIngestion	1.0.0	<span style="color: red;">✖ Broken</span>
danish.GCS.DLRInstallation	1.0.4	<span style="color: green;">☑ Finished</span>
aws.greengrass.clientdevices.mqtt.Moquette	2.0.2	<span style="color: green;">⌚ Running</span>
aws.greengrass.clientdevices.Auth	2.0.4	<span style="color: green;">⌚ Running</span>
danish.GCS.TelemetryIngestion.DevTest	1.0.0	<span style="color: green;">⌚ Installed</span>
danish.GCS.TrashClassification.ModelStore	1.0.0	<span style="color: green;">☑ Finished</span>
aws.greengrass.StreamManager	2.0.14	<span style="color: green;">⌚ Running</span>

## 4.8 Data Ingestion Component Validation

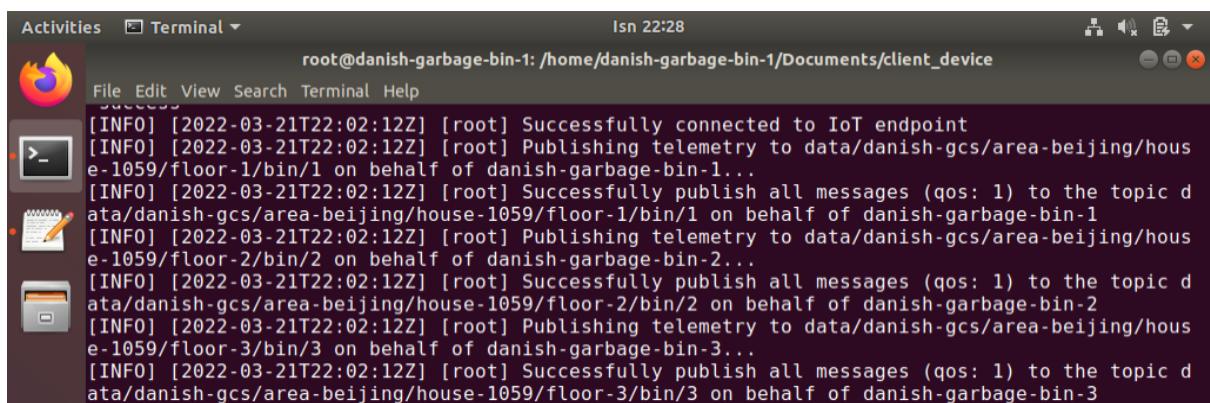
- 1) In the client device, [activate the Python virtual environment](#). Then, run the command below to publish fill level telemetry to the core device and the AWS IoT Core. Refer to the next three subsections to validate the component.

```
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 <i>client_device/pub_telemetry_topics.txt</i> .
message-path	A directory containing files that each contains telemetry records. The sample directory is included in <i>client_device/Telemetry</i> .
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.



The screenshot shows a terminal window titled "Terminal" with the command "lsn 22:28" at the top. The window title bar also includes "Activities" and the path "root@danish-garbage-bin-1: /home/danish-garbage-bin-1/Documents/client\_device". The terminal window displays the following log output:

```
[INFO] [2022-03-21T22:02:12Z] [root] Successfully connected to IoT endpoint
[INFO] [2022-03-21T22:02:12Z] [root] Publishing telemetry to data/danish-gcs/area-beijing/house-1059/floor-1/bin/1 on behalf of danish-garbage-bin-1...
[INFO] [2022-03-21T22:02:12Z] [root] Successfully publish all messages (qos: 1) to the topic data/danish-gcs/area-beijing/house-1059/floor-1/bin/1 on behalf of danish-garbage-bin-1
[INFO] [2022-03-21T22:02:12Z] [root] Publishing telemetry to data/danish-gcs/area-beijing/house-1059/floor-2/bin/2 on behalf of danish-garbage-bin-2...
[INFO] [2022-03-21T22:02:12Z] [root] Successfully publish all messages (qos: 1) to the topic data/danish-gcs/area-beijing/house-1059/floor-2/bin/2 on behalf of danish-garbage-bin-2
[INFO] [2022-03-21T22:02:12Z] [root] Publishing telemetry to data/danish-gcs/area-beijing/house-1059/floor-3/bin/3 on behalf of danish-garbage-bin-3...
[INFO] [2022-03-21T22:02:12Z] [root] Successfully publish all messages (qos: 1) to the topic data/danish-gcs/area-beijing/house-1059/floor-3/bin/3 on behalf of danish-garbage-bin-3
```

#### 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*.

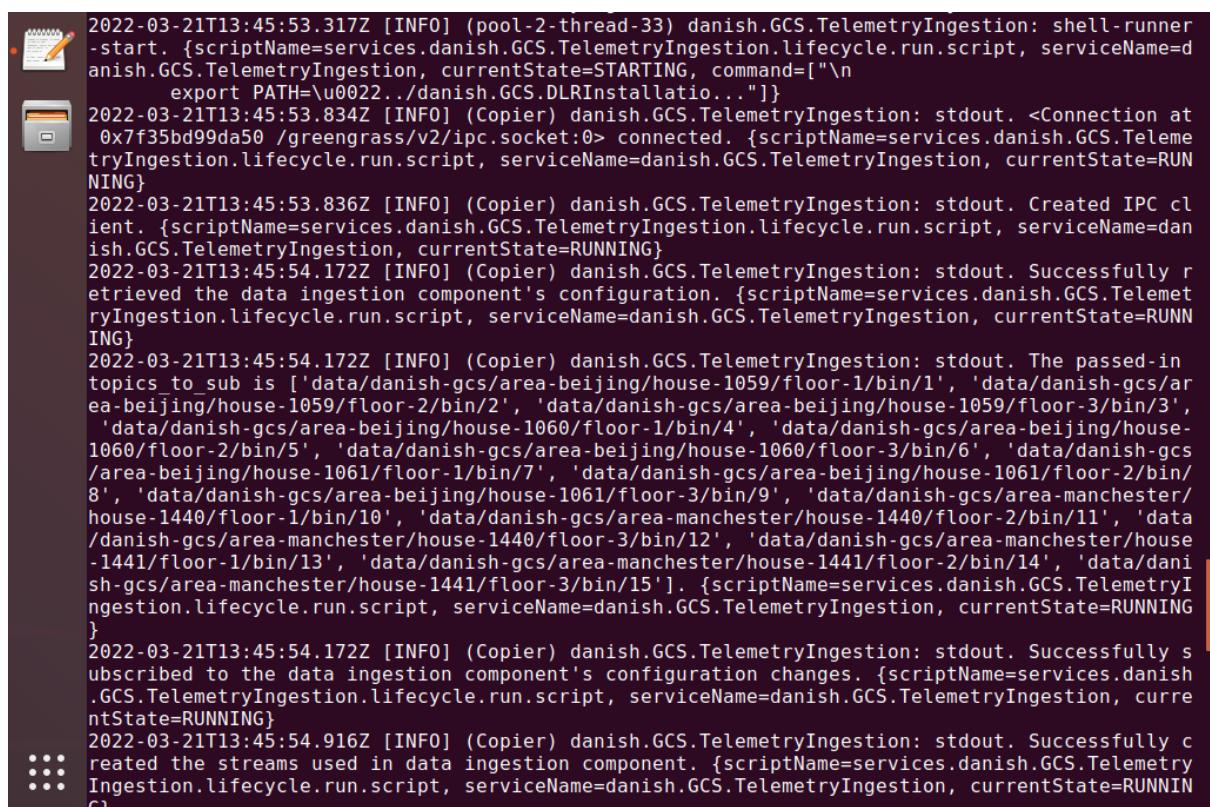
2) Run the command below to view the logs.

```
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.



```
2022-03-21T13:45:53.317Z [INFO] (pool-2-thread-33) danish.GCS.TelemetryIngestion: shell-runner-start. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=STARTING, command=["\n    export PATH=\u0022..\./danish.GCS.DLRInstallatio..."\n]} 2022-03-21T13:45:53.834Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. <Connection at 0x7f35bd99da50 /greengrass/v2/ipc.socket:0> connected. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING} 2022-03-21T13:45:53.836Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. Created IPC client. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING} 2022-03-21T13:45:54.172Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. Successfully retrieved the data ingestion component's configuration. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING} 2022-03-21T13:45:54.172Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. The passed-in topics to_sub is ['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']. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING} 2022-03-21T13:45:54.172Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. Successfully subscribed to the data ingestion component's configuration changes. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING} 2022-03-21T13:45:54.916Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. Successfully created the streams used in data ingestion component. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING}
```

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

```
2022-03-21T13:45:54.920Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. Successfully subscribed to data/danish-gcs/area-beijing/house-1059/floor-1/bin/1. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING}
2022-03-21T13:45:54.921Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. Successfully subscribed to data/danish-gcs/area-beijing/house-1059/floor-2/bin/2. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING}
2022-03-21T13:45:54.921Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. Successfully subscribed to data/danish-gcs/area-beijing/house-1059/floor-3/bin/3. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING}
```

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



```
2022-03-21T14:02:12.530Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. The message received: SubscriptionResponseMessage(binary_message=BinaryMessage(message=b'{"thing_name": "danish-garbage-bin-1", "utc_timestamp": "1647819998", "fill_level": "0.2316", "sequence": "1"}')) . {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING}
2022-03-21T14:02:12.537Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. The message received: SubscriptionResponseMessage(binary_message=BinaryMessage(message=b'{"thing_name": "danish-garbage-bin-1", "utc_timestamp": "1647823598", "fill_level": "0.2533", "sequence": "2"}')) . {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING}
2022-03-21T14:02:12.545Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. The message received: SubscriptionResponseMessage(binary_message=BinaryMessage(message=b'{"thing_name": "danish-garbage-bin-1", "utc_timestamp": "1647827198", "fill_level": "0.4522", "sequence": "3"}')) . {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING}
```

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

```
2022-03-21T14:03:26.464Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. Successfully appended b'{"thing_name": "danish-garbage-bin-15", "utc_timestamp": "1647830798", "fill_level": "0.314", "sequence": "4"}' to FillLevelTelemetryStream with seq no 73. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING}
```

- 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.

```
2022-03-21T14:03:27.481Z [INFO] (Copier) danish.GCS.TelemetryIngestion: stdout. Maximum batch count of 25 has been reached, exporting to FillLevelTelemetryStream.. {scriptName=services.danish.GCS.TelemetryIngestion.lifecycle.run.script, serviceName=danish.GCS.TelemetryIngestion, currentState=RUNNING}
dcs_t8_g35@dcs-202201-t8-g35:~$
```

#### 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/+

The screenshot shows the AWS IoT MQTT Test Client interface. On the left, under 'Subscribe to a topic', the 'Topic filter' is set to 'data/danish-gcs/+/-/+/-/bin/'. On the right, under 'data/danish-gcs/+/-/+/-/bin/+', there are two received messages listed. The first message is from 'data/danish-gcs/area-manchester/house-1441/floor-3/bin/15' at 'March 21, 2022, 22:37:11 (UTC+0800)'. Its payload is a JSON object: { "thing\_name": "danish-garbage-bin-15", "utc\_timestamp": "1647834398", "fill\_level": "0.3348", "sequence": "5" }. The second message is from 'data/danish-gcs/area-manchester/house-1441/floor-3/bin/15' at 'March 21, 2022, 22:37:11 (UTC+0800)'. Its payload is a JSON object: { "thing\_name": "danish-garbage-bin-15", "utc\_timestamp": "1647830798", "fill\_level": "0.314", "sequence": "4" }.

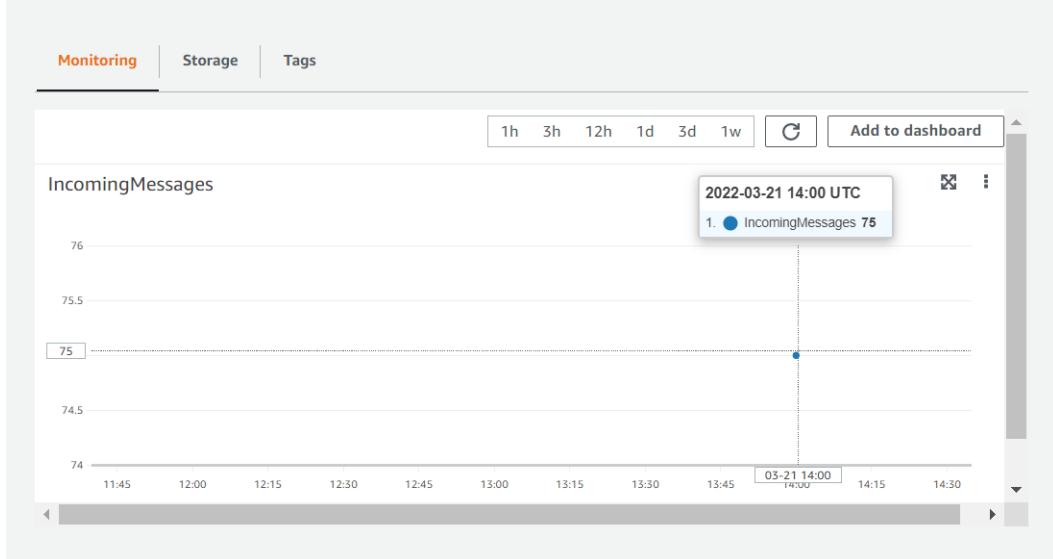
- 2) If the validation above is successful, check that the DynamoDB items has been updated accordingly. In the **DynamoDB Console** navigation menu, select **Tables > DanishGarbageBin > Explore table items**. If not, check the CloudWatch logs for any error.

The screenshot shows the AWS DynamoDB 'DanishGarbageBin' table. At the top, it says 'Completed' and 'Read capacity units consumed: 2'. Below that, it shows 'Items returned (15)'. A table is displayed with columns: ThingName, FillLevel, and LastUpdated. One visible row shows 'danish-garbage-bin-1' with 'FillLevel' 0.2316 and 'LastUpdated' 1647819998.

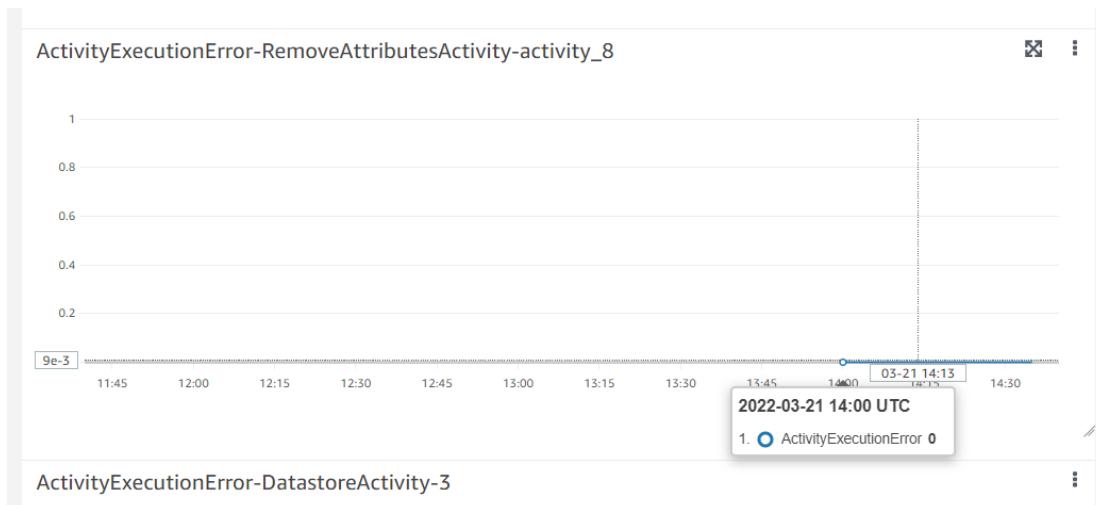
	ThingName	FillLevel	LastUpdated
	danish-garbage-bin-1	0.2316	1647819998

- 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

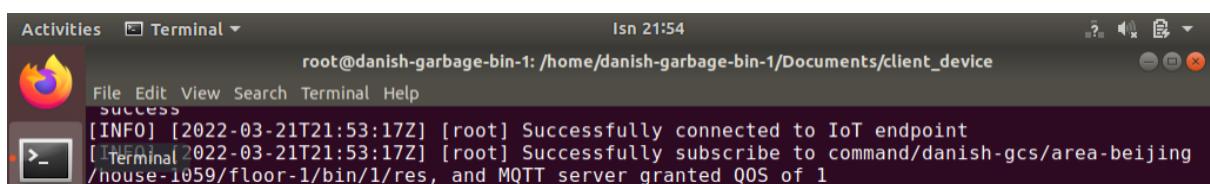
1) In the client device, [activate the Python virtual environment](#). Then, run the command below to request trash classification inference from the core device. Refer to the next three subsections to validate the component.

```
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 <code>python3 ./client_device/create_inference_req.py</code> .
verbosity	The verbosity level used in logging. Possible values are “nologs”, “fatal”, “error”, “warn”, “info”, “debug”, “trace”.

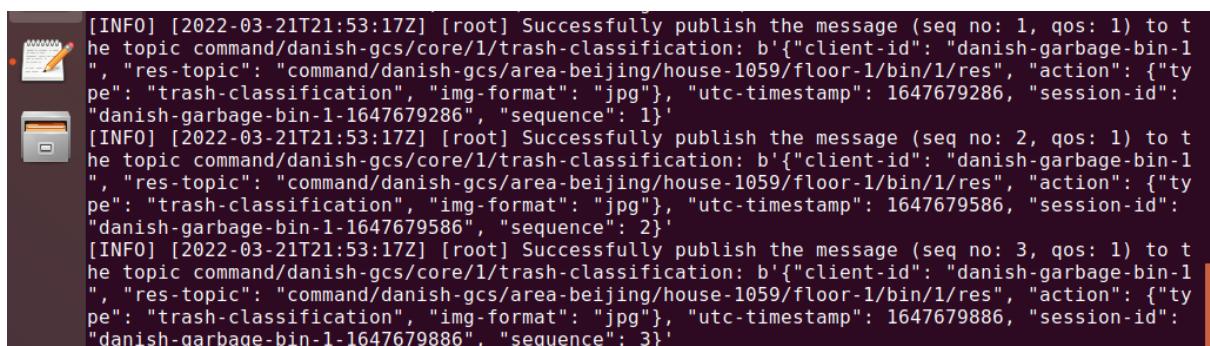
#### 4.9.1 Validate Command's Logs from Garbage Bin/Client Device

- 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.



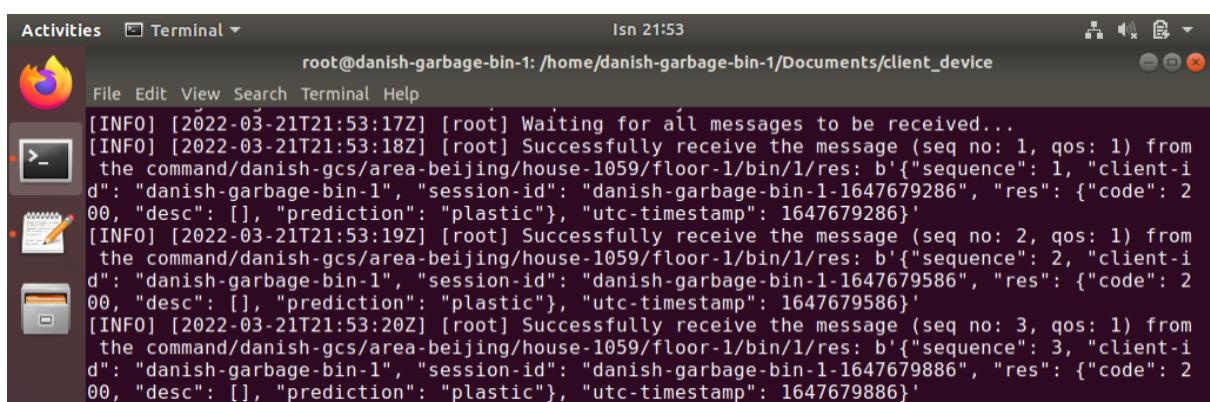
```
root@danish-garbage-bin-1: /home/danish-garbage-bin-1/Documents/client_device
[INFO] [2022-03-21T21:53:17Z] [root] Successfully connected to IoT endpoint
[INFO] [2022-03-21T21:53:17Z] [root] Successfully subscribe to command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res, and MQTT server granted QOS of 1
```

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



```
[INFO] [2022-03-21T21:53:17Z] [root] Successfully publish the message (seq no: 1, qos: 1) to the topic command/danish-gcs/core/1/trash-classification: b'{"client-id": "danish-garbage-bin-1", "res-topic": "command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res", "action": {"type": "trash-classification", "img-format": "jpg"}, "utc-timestamp": 1647679286, "session-id": "danish-garbage-bin-1-1647679286", "sequence": 1}'
[INFO] [2022-03-21T21:53:17Z] [root] Successfully publish the message (seq no: 2, qos: 1) to the topic command/danish-gcs/core/1/trash-classification: b'{"client-id": "danish-garbage-bin-1", "res-topic": "command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res", "action": {"type": "trash-classification", "img-format": "jpg"}, "utc-timestamp": 1647679586, "session-id": "danish-garbage-bin-1-1647679586", "sequence": 2}'
[INFO] [2022-03-21T21:53:17Z] [root] Successfully publish the message (seq no: 3, qos: 1) to the topic command/danish-gcs/core/1/trash-classification: b'{"client-id": "danish-garbage-bin-1", "res-topic": "command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res", "action": {"type": "trash-classification", "img-format": "jpg"}, "utc-timestamp": 1647679886, "session-id": "danish-garbage-bin-1-1647679886", "sequence": 3}'
```

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



```
root@danish-garbage-bin-1: /home/danish-garbage-bin-1/Documents/client_device
[INFO] [2022-03-21T21:53:17Z] [root] Waiting for all messages to be received...
[INFO] [2022-03-21T21:53:18Z] [root] Successfully receive the message (seq no: 1, qos: 1) from the command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res: b'{"sequence": 1, "client-id": "danish-garbage-bin-1", "session-id": "danish-garbage-bin-1-1647679286", "res": {"code": 200, "desc": [], "prediction": "plastic"}, "utc-timestamp": 1647679286}'
[INFO] [2022-03-21T21:53:19Z] [root] Successfully receive the message (seq no: 2, qos: 1) from the command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res: b'{"sequence": 2, "client-id": "danish-garbage-bin-1", "session-id": "danish-garbage-bin-1-1647679586", "res": {"code": 200, "desc": [], "prediction": "plastic"}, "utc-timestamp": 1647679586}'
[INFO] [2022-03-21T21:53:20Z] [root] Successfully receive the message (seq no: 3, qos: 1) from the command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res: b'{"sequence": 3, "client-id": "danish-garbage-bin-1", "session-id": "danish-garbage-bin-1-1647679886", "res": {"code": 200, "desc": [], "prediction": "plastic"}, "utc-timestamp": 1647679886}'
```

#### 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.

```
2022-03-21T13:45:54.505Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Created IPC c
lient. {scriptName=services.danish.GCS.TrashClassification.lifecycle.run.script, serviceName=d
anish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:45:54.589Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully
retrieved the inference component's configuration. {scriptName=services.danish.GCS.TrashClassifi
cation.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNIN
G}
2022-03-21T13:45:54.600Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully
subscribed to the inference component's configuration changes. {scriptName=services.danish.GCS
.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, current
State=RUNNING}
2022-03-21T13:45:55.083Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully
created the streams used in inference component. {scriptName=services.danish.GCS.TrashClassifi
cation.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:45:55.083Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully
subscribed to command/danish-gcs/core/1/trash-classification. {scriptName=services.danish.GCS
.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentS
tate=RUNNING}
```

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.

```
Activities Terminal ▾ Isn 21:56
dcs_t8_g35@dcs-202201-t8-g35: ~

File Edit View Search Terminal Help
2022-03-21T13:53:17.483Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. The message received: SubscriptionResponseMessage(binary_message=BinaryMessage(message=b'{"client-id": "danish-garbage-bin-1", "res-topic": "command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res", "action": {"type": "trash-classification", "img-format": "jpg"}, "utc-timestamp": 1647679286, "session-id": "danish-garbage-bin-1-1647679286", "sequence": 1}')). {scriptName=services.daniFiles@CS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:53:17.796Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Starting to process SubscriptionResponseMessage(binary_message=BinaryMessage(message=b'{"client-id": "danish-garbage-bin-1", "res-topic": "command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res", "action": {"type": "trash-classification", "img-format": "jpg"}, "utc-timestamp": 1647679286, "session-id": "danish-garbage-bin-1-1647679286", "sequence": 1}')). {scriptName=services.daniGCS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:53:17.797Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully validated the inference request and form the response's skeleton: {'sequence': 1, 'client-id': 'danish-garbage-bin-1', 'session-id': 'danish-garbage-bin-1-1647679286', 'res': {'code': 200, 'desc': []}, 'utc-timestamp': 1647679286}. {scriptName=services.danish.GCS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:53:17.899Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully retrieve image from device image/danish-garbage-bin-1-1647679286.jpg. {scriptName=services.danish.GCS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:53:17.899Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Starting to perform inference. {scriptName=services.danish.GCS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:53:18.259Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully performed the inference for request client-id=danish-garbage-bin-1, utc-timestamp=1647679286.. {scriptName=services.danish.GCS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:53:18.259Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. {"client-id": "danish-garbage-bin-1", "utc-timestamp": 1647679286, "img-format": "jpg", "inference-type": "trash-classification", "inference-description": "Top 3 predictions with score 0.5 or above ", "inference-results": []}. {scriptName=services.danish.GCS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:53:18.260Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully published to command/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res: b'{"sequence": 1, "client-id": "danish-garbage-bin-1", "session-id": "danish-garbage-bin-1-1647679286", "res": {"code": 200, "desc": [], "prediction": "plastic"}, "utc-timestamp": 1647679286}'. {scriptName=services.danish.GCS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
```

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.

```
2022-03-21T13:53:37.527Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully uploaded the data (seq no: 0) in the <bound method StatusContext._get_stream_name of <Class StatusContext. s3_export_task_definition: <Class S3Exp...etadata: None>, export_identifier: 'S3 TaskExecut...lOutputStream', stream_name: 'ModelOuputStream', sequence_number: 0>> to S3. {scriptName=services.danish.GCS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
2022-03-21T13:53:38.554Z [INFO] (Copier) danish.GCS.TrashClassification: stdout. Successfully uploaded the data (seq no: 0) in the <bound method StatusContext._get_stream_name of <Class StatusContext. s3_export_task_definition: <Class S3Exp...etadata: None>, export_identifier: 'S3 TaskExecut...hImageStream', stream_name: 'TrashImageStream', sequence_number: 0>> to S3. {scriptName=services.danish.GCS.TrashClassification.lifecycle.run.script, serviceName=danish.GCS.TrashClassification, currentState=RUNNING}
```

#### 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.

Amazon S3 > danish-gcs-image-and-model-output-bucket > images/

Copy S3 URI

images/

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Copy S3 URI  Copy URL  Download  Open  Delete  Actions  Create folder

Upload

Find objects by prefix

< 1 > |

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	images-2022-03-21T21:53:30.zip	zip	March 21, 2022, 21:53:35 (UTC+08:00)	210.5 KB	Standard

Amazon S3 > danish-gcs-image-and-model-output-bucket > model\_outputs/

Copy S3 URI

model\_outputs/

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Copy S3 URI  Copy URL  Download  Open  Delete  Actions  Create folder

Upload

Find objects by prefix

< 1 > |

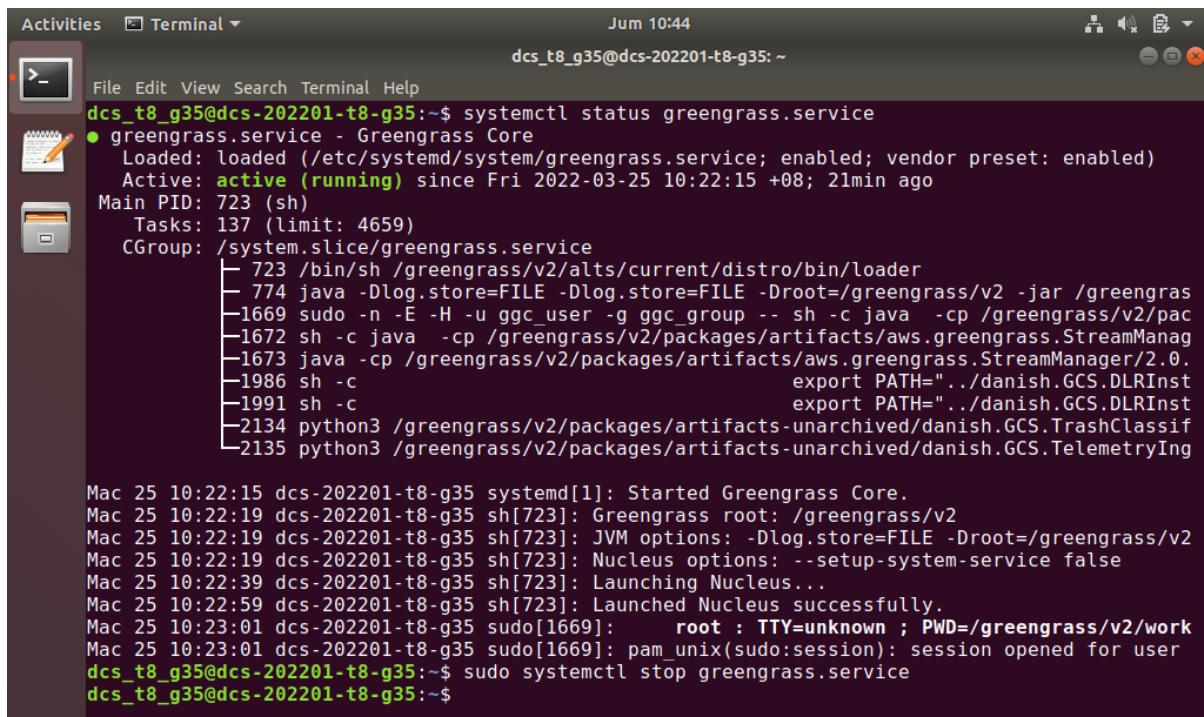
<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	model-outputs-2022-03-21T21:53:30.txt	txt	March 21, 2022, 21:53:35 (UTC+08:00)	3.1 KB	Standard

- 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.

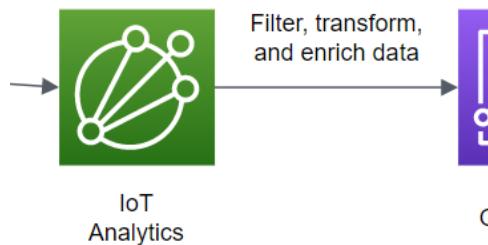
```
sudo systemctl stop greengrass.service
```



```
Activities Terminal Jum 10:44
dcs_t8_g35@dcs-202201-t8-g35:~$ systemctl status greengrass.service
● greengrass.service - Greengrass Core
  Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor preset: enabled)
  Active: active (running) since Fri 2022-03-25 10:22:15 +08; 21min ago
    Main PID: 723 (sh)
      Tasks: 137 (limit: 4659)
     CGroup: /system.slice/greengrass.service
             └─ 723 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
                  ├ 774 java -Dlog.store=FILE -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengras
                  ├ 1669 sudo -n -E -H -u ggc_user -g ggc_group -- sh -c java -cp /greengrass/v2/pac
                  ├ 1672 sh -c java -cp /greengrass/v2/packages/artifacts/aws.greengrass.StreamManag
                  ├ 1673 java -cp /greengrass/v2/packages/artifacts/aws.greengrass.StreamManager/2.0.
                  ├ 1986 sh -c                                     export PATH=../danish.GCS.DLRInst
                  ├ 1991 sh -c                                     export PATH=../danish.GCS.DLRInst
                  ├ 2134 python3 /greengrass/v2/packages/artifacts-unarchived/danish.GCS.TrashClassif
                  ├ 2135 python3 /greengrass/v2/packages/artifacts-unarchived/danish.GCS.TelemetryIng

Mac 25 10:22:15 dcs-202201-t8-g35 systemd[1]: Started Greengrass Core.
Mac 25 10:22:19 dcs-202201-t8-g35 sh[723]: Greengrass root: /greengrass/v2
Mac 25 10:22:19 dcs-202201-t8-g35 sh[723]: JVM options: -Dlog.store=FILE -Droot=/greengrass/v2
Mac 25 10:22:19 dcs-202201-t8-g35 sh[723]: Nucleus options: --setup-system-service false
Mac 25 10:22:39 dcs-202201-t8-g35 sh[723]: Launching Nucleus...
Mac 25 10:22:59 dcs-202201-t8-g35 sh[723]: Launched Nucleus successfully.
Mac 25 10:23:01 dcs-202201-t8-g35 sudo[1669]:      root : TTY=unknown ; PWD=/greengrass/v2/work
Mac 25 10:23:01 dcs-202201-t8-g35 sudo[1669]: pam_unix(sudo:session): session opened for user
dcs_t8_g35@dcs-202201-t8-g35:~$ sudo systemctl stop greengrass.service
dcs_t8_g35@dcs-202201-t8-g35:~$
```

## 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**.

Specify channel details

Details
Channel name Choose a unique name that you can easily identify. <input type="text" value="danish_gcs_bin_telemetry_iot_analytics_channel"/> The channel name must contain 1-128 characters. Valid characters are a-z, A-Z, 0-9, and _ (underscore).

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

Choose storage option

Channel storage
You can use your own S3 bucket or an AWS IoT Analytics managed S3 bucket.
Storage type You can use your own Amazon S3 bucket, which you can access and manage independently. Or, you can use the built-in channel storage in AWS IoT Analytics.
<input type="radio"/> Customer managed storage <input checked="" type="radio"/> Service managed storage
Choose how long to store your raw data By default, AWS IoT Analytics will continue to increase your channel size indefinitely. You can also choose a shorter time period. <input type="text" value="Indefinitely"/>

- 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**.

**Specify data store details**

**Details**

**Data store ID**  
A unique ID identifies your data store. You can't change this ID after you create it.  
  
Valid characters: a-z, A-Z, 0-9, and \_ (underscore).

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

**Configure storage type**

**Data store storage**  
You can use your own S3 bucket or an AWS IoT Analytics managed S3 bucket.

**Storage type**  
You can use your own Amazon S3 bucket, which you can access and manage independently. Or, you can use the built-in data store storage in AWS IoT Analytics.

Customer managed storage  
 Service managed storage

Configure how long you want to keep your processed data  
IoT Analytics will continue to increase the size of your data store indefinitely. You can opt to set a shorter retention policy.

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

**Configure data format**

**Choose data format**  
You can't change the data format after you create the data store.

**Classification**

- 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**.

**Setup pipeline ID and sources**

**Pipeline name and source**

Pipeline name  
danish\_gcs\_bin\_telemetry\_pipeline  
Valid characters: a-z, A-Z, 0-9, and \_ (underscore).

Pipeline source  
The pipeline will read messages from this channel.  
danish\_gcs\_bin\_telemetry\_iot\_analytics\_channel

Pipeline output  
Processed data will be stored in this data store.  
danish\_gcs\_bin\_telemetry\_datastore

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

**Message attributes**

Pipelines enrich, transform, and filter messages based on their attributes. Upload a sample JSON message or enter attributes manually to get started.

**Edit attributes**

Attribute name	Value	Data type	Action
e.g. temperature		String	Add attribute

**Attributes**

Attribute name	Value	Data type	Action
thing_name	"danish-garbage-bin-1"	String	
utc_timestamp	"1647819998"	String	
fill_level	"0.2316"	String	
sequence	"1"	String	

- 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.

▼ Enrich messages with IoT Core registry information

Enrich

**Incoming messages**

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

Enrich messages with IoT Core registry information

Append IoT Core registry metadata to a message as a new attribute.

Attribute name	Source for thing name
device_metadata	thing_name

Select an IAM role that allows IoT Analytics to read IoT Core registry metadata.

Role

DanishGCSDescribeThingRole

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

**Outgoing messages**

Below are the attributes to be included in the outgoing message.

```
{
  "thing_name": "danish-garbage-bin-1",
  "utc_timestamp": "1647819998",
  "fill_level": "0.2316",
  "sequence": "1",
  "device_metadata": {
    "defaultClientId": "danish-garbage-bin-1",
    "thingName": "danish-garbage-bin-1",
    "thingId": "7407030c-303b-49c9-ae9b-44ee1ff4757",
    "thingArn": "arn:aws:iot:us-east-1:385235753061:thing/danish-garbage-bin-1",
    "thingTypeName": "danish-garbage-bin",
    "attributes": {
      "area": "beijing",
      "serialNumber": "ROGUNHQY",
      "manufacturing_day": "20-03-2022",
      "floor": "1",
      "house": "1059",
      "version": "v1"
    },
    "version": 1,
    "billingGroupName": null
  }
}
```

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

**Remove message attributes**  
Determine which attributes will be removed.

<input type="checkbox"/> Remove attribute
<input type="checkbox"/> thing_name
<input type="checkbox"/> utc_timestamp
<input type="checkbox"/> fill_level
<input type="checkbox"/> sequence
<input type="checkbox"/> device_metadata
<input checked="" type="checkbox"/> device_metadata.defaultClientId
<input checked="" type="checkbox"/> device_metadata.thingName
<input checked="" type="checkbox"/> device_metadata.thingId
<input checked="" type="checkbox"/> device_metadata.thingArn
<input checked="" type="checkbox"/> device_metadata.thingTypeName
<input type="checkbox"/> device_metadata.attributes
<input type="checkbox"/> device_metadata.attributes.area
<input checked="" type="checkbox"/> device_metadata.attributes.serialNumber
<input checked="" type="checkbox"/> device_metadata.attributes.manufacturing_day
<input type="checkbox"/> device_metadata.attributes.floor
<input type="checkbox"/> device_metadata.attributes.house
<input checked="" type="checkbox"/> device_metadata.attributes.version
<input checked="" type="checkbox"/> device_metadata.version
<input checked="" type="checkbox"/> device_metadata.billingGroupName

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

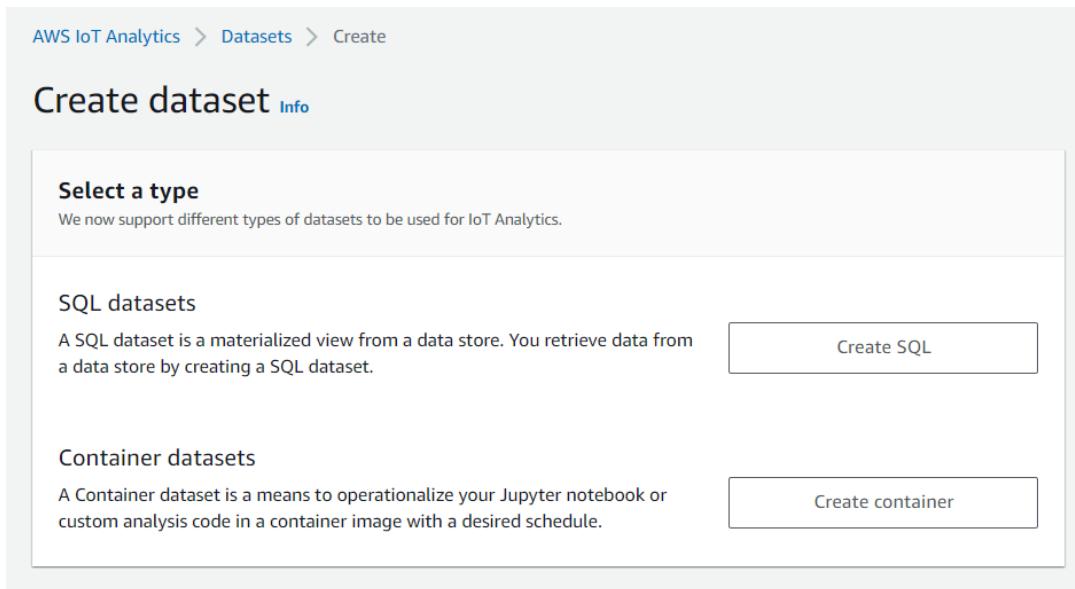
### Outgoing messages

Below are the attributes to be included in the outgoing message.

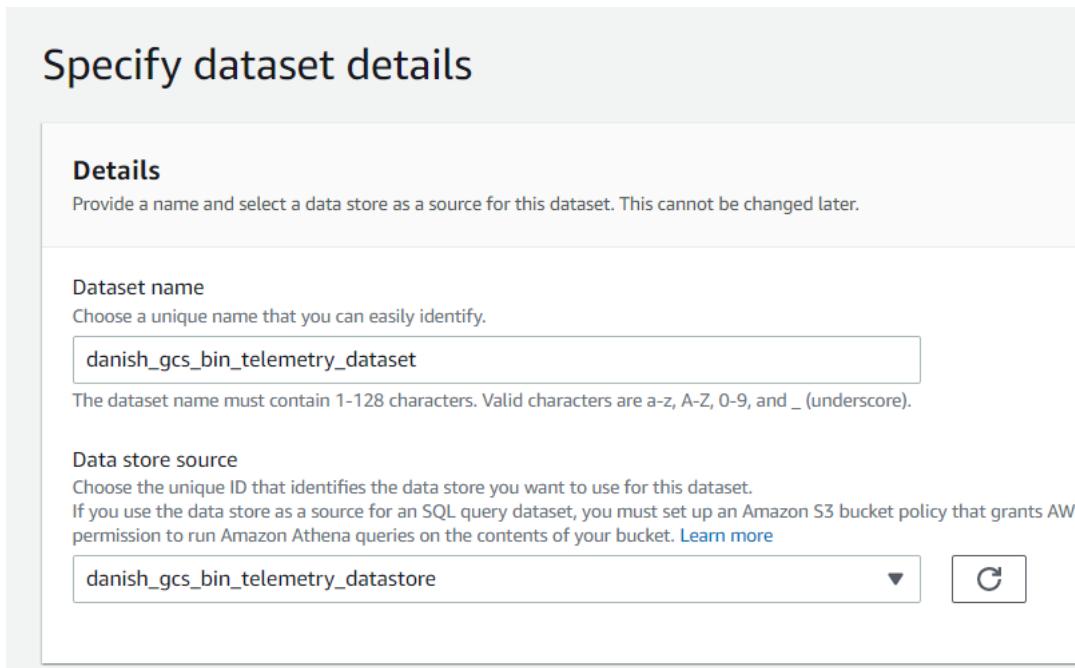
```
{
  "thing_name": "danish-garbage-bin-1",
  "utc_timestamp": "1647819998",
  "fill_level": "0.2316",
  "sequence": "1",
  "device_metadata": {
    "attributes": {
      "area": "beijing",
      "floor": "1",
      "house": "1059"
    }
  }
}
```

## 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 **danic\_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

```

## Author SQL query

### Author query

Enter a SQL query that uses a wildcard to select all attributes and values from your data store.

```

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

```

[Test query](#)

▼ Result preview

thing_name	timestamp	fill_level	house_area	house_number
danish-garbage-bin-10	2022-03-20 23:46:38.000	0.338	manchester	1440
danish-garbage-bin-2	2022-03-20 23:46:38.000	0.0157	beijing	1059
danish-garbage-bin-5	2022-03-20 23:46:38.000	0.2995	beijing	1060

▼ Result preview

	timestamp	fill_level	house_area	house_number	floor_number
e-bin-10	2022-03-20 23:46:38.000	0.338	manchester	1440	1
e-bin-2	2022-03-20 23:46:38.000	0.0157	beijing	1059	2
e-bin-5	2022-03-20 23:46:38.000	0.2995	beijing	1060	2

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**.

**Set query schedule - optional**

**Set query schedule**

Schedule the query to run regularly to refresh the dataset. Please note that the old dataset will be overwritten each time the query is run.

Frequency  
Dataset schedules can be created and edited at any time.

Hourly ▾

Minute of hour  
Starts after this offset

0

The minute of hour can be 0-59 minutes.

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

**Configure the results of your dataset**

**Configure the results of your dataset**

Configure how long you want to keep your results  
IoT Analytics will create versions of this dataset content and store your analytics results for the specified period. We recommend 90 days, however you can opt to set your custom retention policy. You may also limit the number of stored versions of your dataset content.

Indefinitely ▾

Versioning

Disabled ▾

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

8) Select the **danic\_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.

## danish\_gcs\_bin\_telemetry\_dataset

[Run now](#)[Delete](#)

## Overview

## Dataset ARN Info

arn:aws:iotanalytics:us-east-1:385235753061:dataset/danish\_gcs\_bin\_telemetry\_dataset

## Type

CloudWatch Metrics

## Created

Mar 21, 2022 4:49:38 PM +0800

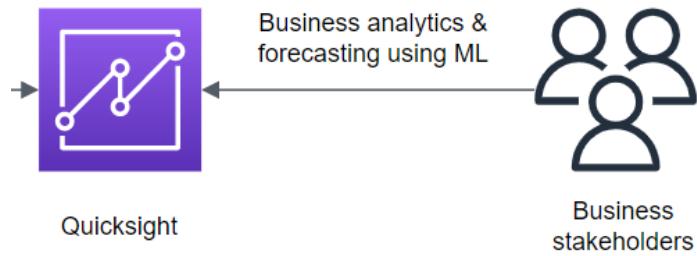
## Last updated

Mar 21, 2022 4:49:38 PM +0800

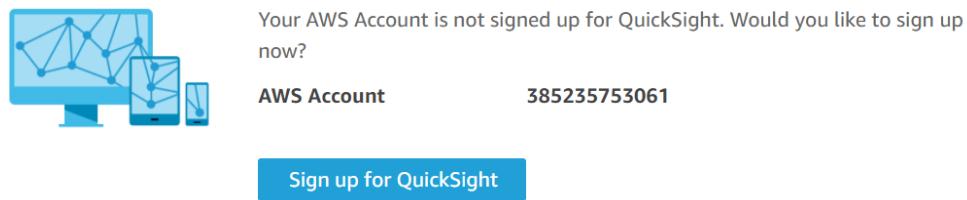
Dataset contents (1)					
	Date	Name	Status	Duration	Actions
<input type="checkbox"/>	Mar 22, 2022 1:00:00 PM +0800	f36550a2-b106-44eb-a79a-8c5ecab2a7ba	<span>✓ Succeeded</span>	1750 ms	

Result preview					
bb7fa3cc-909b-46be-b428-d773b40976ce.csv					
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.

A screenshot of the "Create your QuickSight account" page showing the "Edition" selection. The "Standard" edition is selected, indicated by a blue border around the "FREE" column. The table compares four editions: Team trial for 30 days (4 authors)\*, Author per month (yearly)\*\*, Author per month (monthly)\*\*, and Readers (pay-per-Session). The "Enterprise" edition is also listed for reference.

Edition	<input checked="" type="radio"/> Enterprise	<input type="radio"/> Enterprise + Q Learn more
Team trial for 30 days (4 authors)*	<b>FREE</b>	<b>FREE</b>
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)****

Create your QuickSight account

**Standard**

**Authentication method**

- Use IAM federated identities & QuickSight-managed users
  - Authenticate with single sign-on (SAML or OpenID Connect), AWS IAM credentials, or QuickSight credentials
- Use IAM federated identities only
  - Authenticate with single sign-on (SAML or OpenID Connect) or AWS IAM credentials

## QuickSight region

Select a region



US East (N. Virginia)



## Account info

### QuickSight account name

You will need this for you and others to sign in



DCS-202201-T8-G35

### Notification email address

For QuickSight to send important notifications

polarbearayap2@gmail.com

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

## QuickSight access to AWS services

Make your existing AWS data and users available in QuickSight. [Learn more](#)

### Allow access and autodiscovery for these resources

Amazon Redshift

Amazon RDS

IAM

Amazon S3  
[Select S3 buckets](#)

Amazon Athena

Make sure you've chosen the right Amazon S3 buckets for QuickSight access

Amazon S3 Storage Analytics

AWS IoT Analytics

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

Congratulations! You are signed up for Amazon QuickSight!

Access QuickSight with the following information

Account name: dcs-202201-t8-g35

Username: DCS\_202201\_T8\_G35

[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.

The image consists of three vertically stacked screenshots of the QuickSight web interface, illustrating the process of creating a new dataset.

**Screenshot 1: Datasets Overview**  
The top screenshot shows the main QuickSight dashboard under the "Datasets" tab. A sidebar on the left includes sections for Favorites, Recent, Dashboards, Analyses, and Datasets (which is selected). On the right, a list of existing datasets is displayed:

Name	Owner	Last Modified
Business Review	SPICE	7 minutes ago
People Overview	SPICE	7 minutes ago
Sales Pipeline	SPICE	7 minutes ago
Web and Social Media Analytics	SPICE	7 minutes ago

**Screenshot 2: New AWS IoT Analytics Data Source Creation**  
The middle screenshot shows a modal dialog titled "New AWS IoT Analytics data source". It contains fields for "Data source name" (with placeholder "Enter a name for the data source") and "Select an AWS IoT Analytics dataset to import:" (with an option for "danish\_gcs\_bin\_telemetry\_dataset"). At the bottom are "Cancel" and "Create data source" buttons.

**Screenshot 3: Finish Dataset Creation Confirmation**  
The bottom screenshot shows another modal dialog titled "Finish dataset creation". It displays summary information: "Table: danish\_gcs\_bin\_telemetry\_dataset", "Estimated table size... 11.9KB", "Data source: danish\_gcs\_bin\_telemetry\_dataset", and "Import to SPICE ✓ 1009.5MB available". It also features "Edit/Preview data" and "Visualize" buttons.

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

The screenshot shows the QuickSight interface with the following details:

- Header:** QuickSight, DCS\_202201\_T8\_G35
- Toolbar:** Add, Undo, Redo, danish\_gcs\_bin\_telemetry\_dataset..., Autosave On, Save as, Export, Share
- Left Sidebar:**
  - Visualize:** Dataset (SPICE), Fields list (Search fields), AutoGraph (Choose 1 or more fields and let QuickSight choose the most appropriate chart).
  - Filter:**
  - Parameters:**
  - Actions:**
  - Themes:**
  - Settings:**
- Field wells:** Sheet 1
- Import complete:** 100% success, 75 rows were imported to SPICE, 0 rows were skipped.

## 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.DLRIImageClassification.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=\\"$({danish.GCS.DLRInstallation:configuration:/MLRootPath})/\n        greengrass_ml_dlr_conda/\n        bin:$PATH\"\n        eval \"$(${danish.\n        GCS.DLRInstallation:configuration:/MLRootPath})/\n        greengrass_ml_dlr_conda/bin/conda shell.bash hook)\n        \"\n        conda activate\n        greengrass_ml_dlr_conda\n        {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.

```
j,
"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.

```
Manual Configuration > machine-learning-component > vi:  
1   name: greengrass_ml_dlr_conda  
2   channels:  
3     - defaults  
4   dependencies:  
5     - pip  
6     - numpy  
7     - opencv  
8     - pip:  
9       - Pillow>=8.4.0  
10      - scipy>=1.5.4  
11      - awsiotsdk>=1.9.2  
12      - stream-manager>=1.1.1
```

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 recipies 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": {
            "policyDescription": "Allow the component to receive inference requests from the client devices.",
            "operations": [
                "aws.greengrass#SubscribeRequest"
            ],
            "resources": [
                "inference/danish-gcs/area-beijing/house-1059/floor-1/bin/1/res",
                "inference/danish-gcs/area-beijing/house-1059/floor-2/bin/2/res",
                "inference/danish-gcs/area-beijing/house-1059/floor-3/bin/3/res",
                "inference/danish-gcs/area-beijing/house-1060/floor-1/bin/4/res",
                "inference/danish-gcs/area-beijing/house-1060/floor-2/bin/5/res",
                "inference/danish-gcs/area-beijing/house-1060/floor-3/bin/6/res",
                "inference/danish-gcs/area-beijing/house-1061/floor-1/bin/7/res",
                "inference/danish-gcs/area-beijing/house-1061/floor-2/bin/8/res",
                "inference/danish-gcs/area-beijing/house-1061/floor-3/bin/9/res",
                "inference/danish-gcs/area-manchester/house-1440/floor-1/bin/10/res",
                "inference/danish-gcs/area-manchester/house-1440/floor-2/bin/11/res",
                "inference/danish-gcs/area-manchester/house-1440/floor-3/bin/12/res",
                "inference/danish-gcs/area-manchester/house-1441/floor-1/bin/13/res",
                "inference/danish-gcs/area-manchester/house-1441/floor-2/bin/14/res",
                "inference/danish-gcs/area-manchester/house-1441/floor-3/bin/15/res"
            ]
        }
    }
}

```

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

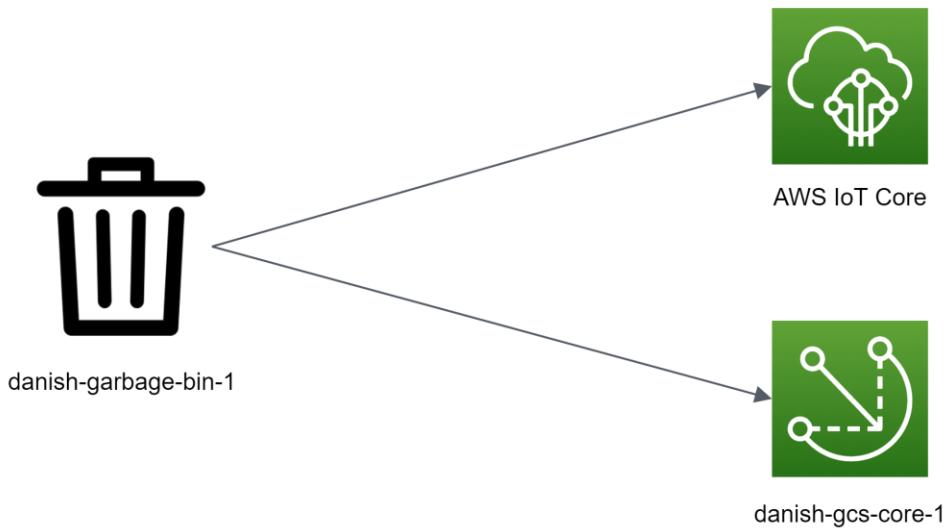
Below are the Python packages that are required in the data ingestion component:

- 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.

```
Manual Configuration > machine-learning-component > v:  
1   name: greengrass_ml_dlr_conda  
2   channels:  
3     - defaults  
4   dependencies:  
5     - pip  
6     - numpy  
7     - opencv  
8     - pip:  
9       - Pillow>=8.4.0  
10      - scipy>=1.5.4  
11      - awsiotsdk>=1.9.2  
12      - stream-manager>=1.1.1
```

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/+/*/*/*/*",
        "source": "LocalMqtt",
        "target": "IotCore"
    },
    "BinToStreamManagerClientComponentMapping": {
        "topic": "data/danish-gcs/+/*/*/*/*",
        "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](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](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].