



Edge Computing with AWS IoT Greengrass

Çağatay Sönmez
1 November 2024

The Primary Sources Used in This Lecture

- <https://docs.aws.amazon.com/greengrass/v2/developerguide>
- <https://docs.aws.amazon.com/greengrass/v2/developerguide/setting-up.html>
- <https://docs.aws.amazon.com/greengrass/v2/developerguide/develop-greengrass-components.html>
- <https://docs.aws.amazon.com/greengrass/v2/developerguide/ip-detector-component.html>
- <https://docs.aws.amazon.com/greengrass/v2/developerguide/interprocess-communication.html>
- <https://docs.aws.amazon.com/greengrass/v2/developerguide/run-lambda-functions.html>
- <https://docs.aws.amazon.com/greengrass/v2/developerguide/interact-with-local-iot-devices.html>
- <https://docs.aws.amazon.com/greengrass/v2/developerguide/client-devices-tutorial.html>
- All the source codes used in this Lecture can be found in the following GitHub repo:
 - <https://github.com/CagataySonmez/AWS-IoT-Greengrass-Demo>

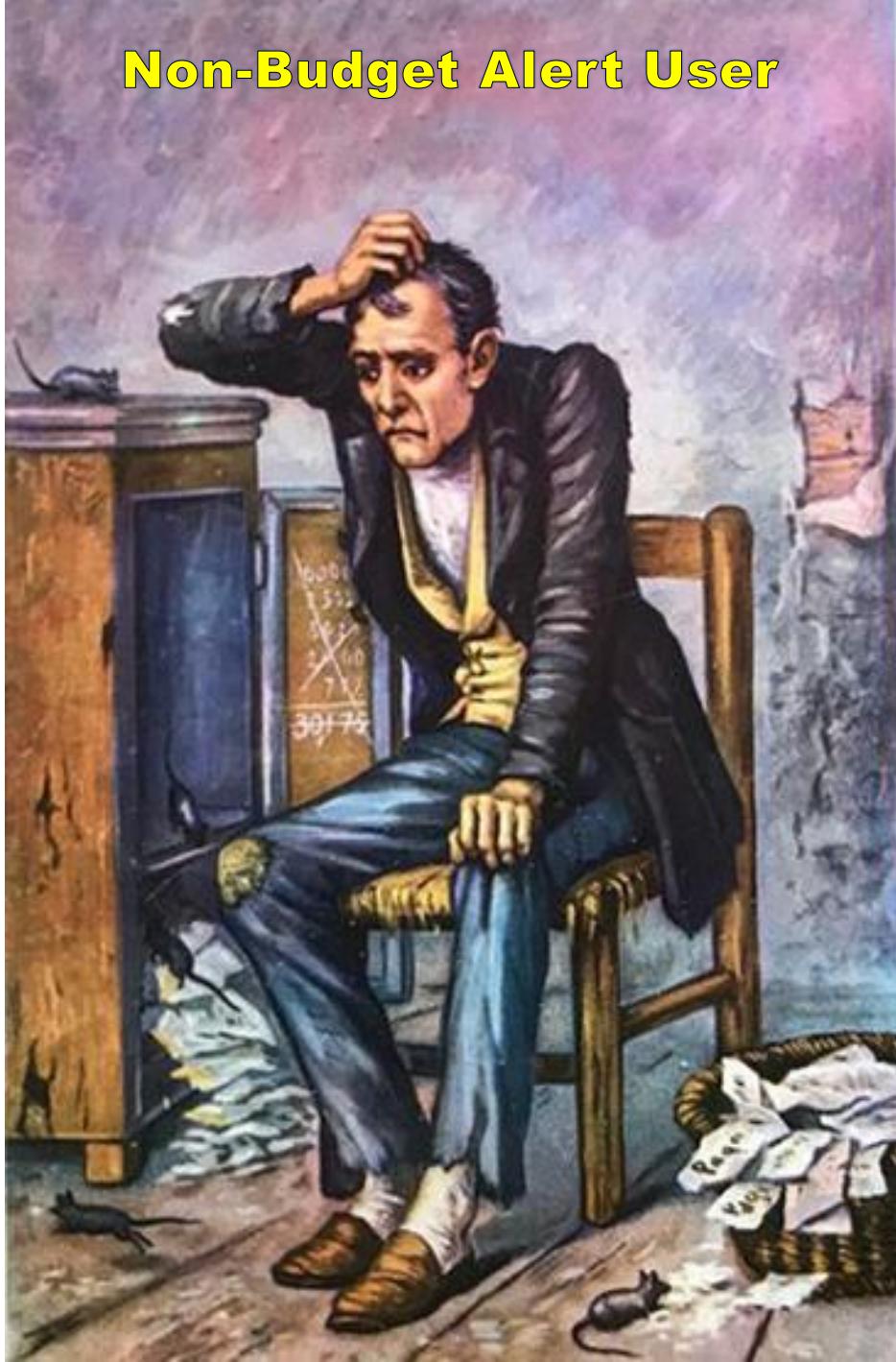
⚠ Reminder

Monitor AWS Free Tier usage via **Budget Alerts** to avoid unexpected charges!

To ensure you don't exceed the Free Tier limits additional costs, please set up budget alerts within your AWS account.

- ✓ **Go to AWS Billing Dashboard** – You can find this under your account settings.
- ✓ **Create a Budget** – Set a budget for \$0 or a minimal amount to receive alerts as soon as usage might incur charges.
- ✓ **Configure Alerts** – Enable notifications so you'll get an email if your usage approaches your set limit.

Non-Budget Alert User

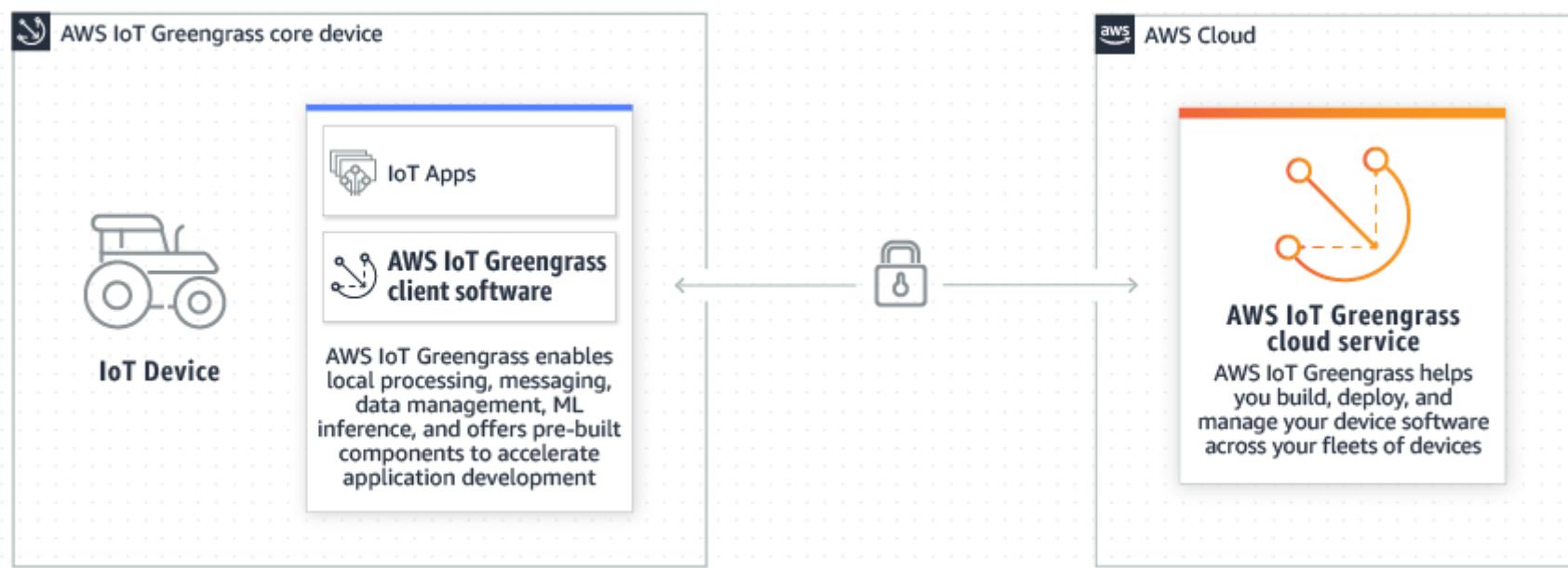


Budget Alert User



What is AWS IoT Greengrass?

- AWS IoT Greengrass is software that extends cloud capabilities to local devices. This enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. Local devices can also communicate securely with AWS IoT Core and export IoT data to the AWS Cloud¹.

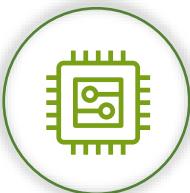


¹ <https://docs.aws.amazon.com/greengrass/v2/developerguide/what-is-iot-greengrass.html>

AWS IoT Greengrass Key Concepts



Greengrass Core Device: A device that runs the AWS IoT Greengrass Core software. A Greengrass core device is an AWS IoT thing. You can add multiple core devices to AWS IoT thing groups. This could be a Raspberry Pi, an industrial PC, or any other compatible device.



Greengrass Client Device: Any device (e.g. an industrial sensor) that communicates with a Greengrass Core device over MQTT. Client devices, which could be an IoT device, a mobile app, or any other system, use the *AWS IoT Device SDK* to interact with Greengrass Core devices.



Greengrass Core Software: The set of all AWS IoT Greengrass software that you install on a core device. AWS IoT Greengrass Core software comprises the Nucleus and other components. It is provided by AWS that enables a device to connect to AWS IoT Core.

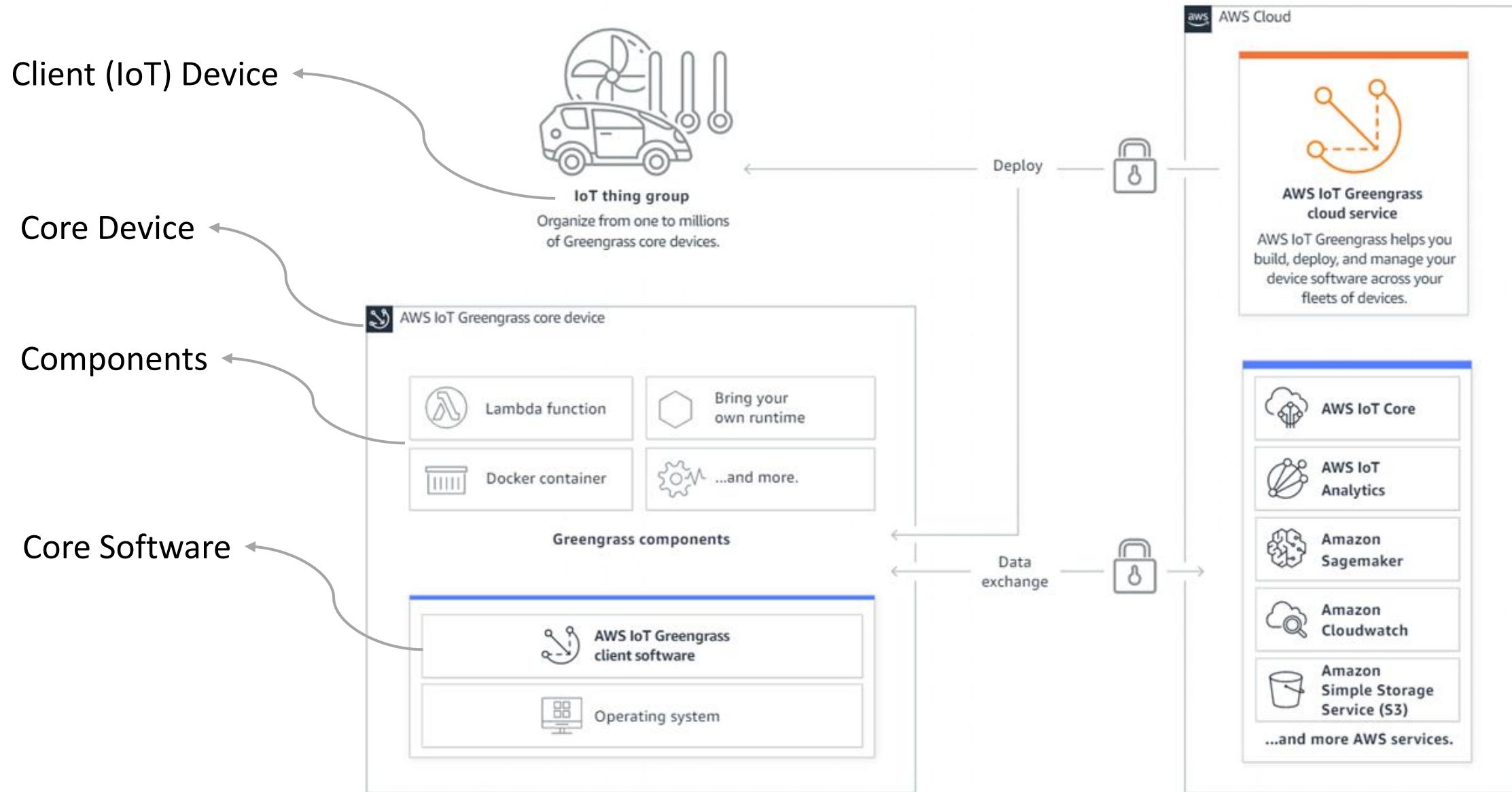


Greengrass Component: A software module (e.g. Python script) that is deployed to and runs on a Greengrass core device. These modules are modeled as components which can perform various tasks, such as collecting data from sensors, controlling actuators, or running machine learning models.



Deployment: The process of installing and configuring Greengrass components on a Greengrass Core device. This typically involves uploading the component code and configuration files to the device.

How AWS IoT Greengrass works



Selecting Client's Operating System

- Some features are supported on only certain operating systems!

| Security | | Linux | Windows |
|--|--|--|---|
| Feature | | | |
| Use a hardware security module (HSM) to securely store the device's private key and certificate | | | <input checked="" type="checkbox"/> Yes <input type="checkbox"/> No |
| Component features | | | |
| Feature | | | |
| Deploy and invoke Lambda functions | | | |
| Publish messages to Amazon Simple Notification Service using the Amazon SNS component | | | |
| Publish data to Amazon Kinesis Data Firehose delivery streams using the Kinesis Data Firehose component | | | |
| Publish video streams to Amazon Kinesis Video Streams using the edge connector for Kinesis Video Streams component | | | |
| Run AWS IoT Greengrass in a Docker container using a prebuilt Docker image | | Linux <input checked="" type="checkbox"/> Yes | Windows <input type="checkbox"/> No |

| Remote maintenance and updates | | Linux | Windows |
|--|--|-------|---|
| Feature | | | |
| Manage core devices with AWS Systems Manager | | | <input checked="" type="checkbox"/> Yes <input type="checkbox"/> No |
| Connect to core devices with AWS IoT secure tunneling | | | <input checked="" type="checkbox"/> Yes <input type="checkbox"/> No |
| Machine learning | | | |
| Feature | | | |
| Perform machine learning inference using Amazon Lookout for Vision | | | <input checked="" type="checkbox"/> Yes <input type="checkbox"/> No |

AWS IoT Greengrass Core Software

AWS IoT Greengrass Core Software

- The AWS IoT Greengrass Core software extends AWS functionality onto an AWS IoT Greengrass core device
- The AWS IoT Greengrass Core software provides a lot of functionality:
 - Process data streams locally with automatic exports to the AWS Cloud.
 - Support MQTT messaging between AWS IoT and components.
 - Interact with local devices that connect and communicate over MQTT.
 - Support local publish and subscribe messaging between components.
 - Deploy and invoke components and Lambda functions.
 - Perform secure, over-the-air (OTA) software updates.
 - Provide secure, encrypted storage of local secrets.
 - Secure connection between devices and AWS Cloud with device authentication and authorization.
 - And more...

Step 1: Set Up Your Environment

For Linux Based Devices

- AWS IoT Greengrass Core software requires Java runtime.
- Amazon recommends to use OpenJDK 11 or Amazon Corretto 11.

For Debian-based or Ubuntu-based distributions:

```
sudo apt install default-jdk
```

For Red Hat-based distributions:

```
sudo yum install java-11-openjdk-devel
```

Step 2: Create AWS Credentials

Create a Temporary User

- Greengrass core software installer uses the security credentials to make programmatic requests for AWS resources.
- We use a temporary user to install code software.
- We will remove this user after the installation is completed.

IAM > Users > Create user

Step 1 Specify user details

Step 2 Set permissions

Step 3 Review and create

Specify user details

User details

User name

CMPE583-GreenGrassUser

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a [best practice](#) to manage their access in IAM Identity Center.

Info If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user.
[Learn more](#)

Cancel Next

Step 2: Create AWS Credentials

Create Access Keys for the User

The screenshot shows the AWS IAM User Security Credentials page for the user 'CMPE583-GreenGrassUser'. The navigation path is IAM > Users > CMPE583-GreenGrassUser. The 'Security credentials' tab is selected. The 'Access keys' section displays a message: 'No access keys. As a best practice, avoid using long-term credentials like access keys instead, use tools which provide short term credentials.' A large orange arrow points from this message down to the 'Create access key' button.

IAM > Users > CMPE583-GreenGrassUser

CMPE583-GreenGrassUser [Info](#)

[Delete](#)

Permissions Groups Tags **Security credentials** Access Advisor

Console sign-in

Enable console access

Console sign-in link <https://arcelik-electronics.signin.aws.amazon.com/console>

Console password Not enabled

Access keys (0)

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. [Learn more](#)

[Create access key](#)

No access keys. As a best practice, avoid using long-term credentials like access keys instead, use tools which provide short term credentials. [Learn more](#)

[Create access key](#)

Step 2: Create AWS Credentials

Copy the Access Keys

The screenshot shows the AWS IAM 'Create access key' page. The navigation path is IAM > Users > CMPE583-GreenGrassUser > Create access key. On the left, there are three steps: Step 1 (Access key best practices & alternatives), Step 2 - optional (Set description tag), and Step 3 (Retrieve access keys). Step 3 is currently selected. The main content area is titled 'Retrieve access keys' with an 'Info' link. It contains a table with two columns: 'Access key' and 'Secret access key'. The 'Access key' column shows a placeholder icon and the value AKIAWY5JNGIBCSFIMYHZ. The 'Secret access key' column shows a placeholder icon and a redacted value followed by a 'Show' link. Below this, a section titled 'Access key best practices' lists four bullet points: 'Never store your access key in plain text, in a code repository, or in code.', 'Disable or delete access key when no longer needed.', 'Enable least-privilege permissions.', and 'Rotate access keys regularly.' A note at the bottom of this section points to 'best practices for managing AWS access keys'. At the bottom right are 'Download .csv file' and 'Done' buttons.

IAM > Users > CMPE583-GreenGrassUser > Create access key

Step 1
[Access key best practices & alternatives](#)

Step 2 - optional
[Set description tag](#)

Step 3
Retrieve access keys

Retrieve access keys [Info](#)

| Access key | Secret access key |
|---|---|
| <input type="text"/> AKIAWY5JNGIBCSFIMYHZ | <input type="text"/> ***** Show |

Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

[Download .csv file](#) [Done](#)

Step 3: Set Up a Greengrass Core Device in AWS

The screenshot shows the AWS IoT Greengrass Core devices page. The left sidebar has a 'Core devices' section selected. The main area displays 'Greengrass core devices (0)' with a search bar and buttons for 'Configure cloud discovery' and 'Set up one core device'. An orange arrow points to the 'Set up one core device' button. Below the table, it says 'No core devices' and 'You don't have any Greengrass core devices in eu-central-1.' A large orange arrow also points to the 'Set up one core device' button.

AWS IoT > Greengrass > Core devices

Greengrass core devices Info

Greengrass core devices (0) C Configure cloud discovery Set up one core device

Search by core device name

| Name | Status | Status reported |
|---|--------|-----------------|
| No core devices | | |
| You don't have any Greengrass core devices in eu-central-1. | | |
| Set up one core device | | |

Manage

- All devices
- Greengrass devices

Core devices

- Components
- Deployments
- Groups (V1)
- LPWAN devices
- Software packages Preview
- Remote actions
- Message routing
- Retained messages
- Security
- Fleet Hub

Device software

Billing groups

Settings

Feature spotlight

Step 3: Set Up a Greengrass Core Device in AWS

The screenshot shows the AWS IoT Greengrass Core Device setup process. It consists of two main sections: Step 1: Register a Greengrass core device and Step 2: Add to a thing group to apply a continuous deployment.

Step 1: Register a Greengrass core device
Greengrass core devices are AWS IoT things. Enter a thing name to be used to create a Greengrass core device.

Core device name
The name of the AWS IoT thing to create. We generated the following name for you.
CMPE583-GreengrassCore
The name can be up to 128 characters. Valid characters: a-z, A-Z, 0-9, underscore (_), and hyphen (-).

Step 2: Add to a thing group to apply a continuous deployment
Add your Greengrass core device to an AWS IoT thing group. If the thing group has an active Greengrass deployment, your new core device receives and applies the deployment when you finish the setup process. To deploy to only the core device, select No group.

Thing group
 Enter a new group name
 Select an existing group
 No group

Thing group name
The name of the AWS IoT thing group to create.
CMPE583-GreengrassGroup
The name can be up to 128 characters. Valid characters: a-z, A-Z, 0-9, underscore (_), and hyphen (-).

The name of the AWS IoT thing for your Greengrass core device.

The name of the new group to create to apply a continuous deployment.

Step 4: Install the Greengrass Core Software

Configure AWS Credentials on the Client

- The Greengrass installer uses AWS credentials to provision the AWS resources that it requires.

Provide credentials as environment variables before running the installer:

```
export AWS_ACCESS_KEY_ID=<AWS_ACCESS_KEY_ID>
export AWS_SECRET_ACCESS_KEY=<AWS_SECRET_ACCESS_KEY>
export AWS_SESSION_TOKEN=<AWS_SESSION_TOKEN>
```

- Credentials are not saved by the Greengrass installer!

Step 4: Install the Greengrass Core Software

For Linux Based Devices

- AWS IoT provides an installer that you can use to set up a Greengrass core device.
- The installer provisions the Greengrass core device as an AWS IoT thing and connects the device to AWS IoT.

Download the installer:

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass.zip &&  
unzip greengrass.zip -d GreengrassInstaller
```

Run the installer:

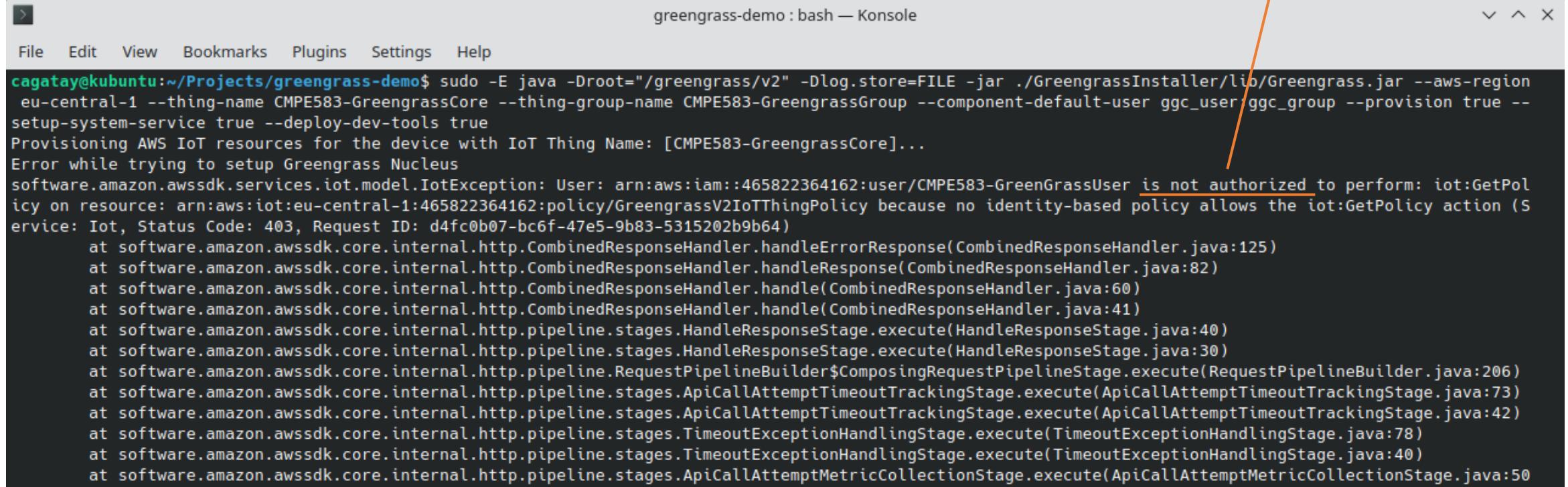
```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE -jar ./GreengrassInstaller/lib/Greengrass.jar --aws-  
region eu-central-1 --thing-name CMPE583-GreengrassCore --thing-group-name CMPE583-  
GreengrassGroup --component-default-user ggc_user:ggc_group --provision true --setup-system-service  
true --deploy-dev-tools true
```

★ Up-to-date installer commands are provided in AWS Console while creating a core device!

Step 4: Install the Greengrass Core Software

Run the Installer Command

The user is not authorized to perform installer jobs! You should attach required permissions!



A screenshot of a terminal window titled "greengrass-demo : bash — Konsole". The window has standard Linux-style menu options: File, Edit, View, Bookmarks, Plugins, Settings, Help. The terminal output shows a command being run:

```
cagatay@kubuntu:~/Projects/greengrass-demo$ sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE -jar ./GreengrassInstaller/lib/Greengrass.jar --aws-region eu-central-1 --thing-name CMPE583-GreengrassCore --thing-group-name CMPE583-GreengrassGroup --component-default-user ggc_user/ggc_group --provision true --setup-system-service true --deploy-dev-tools true
```

The command is provisioning AWS IoT resources for a device with IoT Thing Name: [CMPE583-GreengrassCore]... However, it fails with the following error message:

```
Error while trying to setup Greengrass Nucleus
software.amazon.awssdk.services.iot.model.IotException: User: arn:aws:iam::465822364162:user/CMPE583-GreengrassUser is not authorized to perform: iot:GetPolicy on resource: arn:aws:iot:eu-central-1:465822364162:policy/GreengrassV2IoTThingPolicy because no identity-based policy allows the iot:GetPolicy action (Service: Iot, Status Code: 403, Request ID: d4fc0b07-bc6f-47e5-9b83-5315202b9b64)
    at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handleErrorResponse(CombinedResponseHandler.java:125)
    at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handleResponse(CombinedResponseHandler.java:82)
    at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handle(CombinedResponseHandler.java:60)
    at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handle(CombinedResponseHandler.java:41)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.HandleResponseStage.execute(HandleResponseStage.java:40)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.HandleResponseStage.execute(HandleResponseStage.java:30)
    at software.amazon.awssdk.core.internal.http.pipeline.RequestPipelineBuilder$ComposingRequestPipelineStage.execute(RequestPipelineBuilder.java:206)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptTimeoutTrackingStage.execute(ApiCallAttemptTimeoutTrackingStage.java:73)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptTimeoutTrackingStage.execute(ApiCallAttemptTimeoutTrackingStage.java:42)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.TimeoutExceptionHandlingStage.execute(TimeoutExceptionHandlingStage.java:78)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.TimeoutExceptionHandlingStage.execute(TimeoutExceptionHandlingStage.java:40)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptMetricCollectionStage.execute(ApiCallAttemptMetricCollectionStage.java:50)
```

An orange arrow points from the text "The user is not authorized to perform installer jobs! You should attach required permissions!" to the error message in the terminal.

Step 4: Install the Greengrass Core Software

Set Required Permissions to Greengrass User

- When installing a core device, the user we created needs certain permissions to provision AWS IoT resources.

Permission content available in GitHub
• [AWS/IAM/cmpe583-GreengrassUser-ProvisionPolicy.json](#)

The screenshot shows the AWS IAM Policy Editor interface. On the left, there's a sidebar with 'Step 1 Specify permissions' and 'Step 2 Review and create'. The main area is titled 'Specify permissions' with a 'Info' link. It contains a 'Policy editor' section where a JSON policy document is displayed. The JSON code is as follows:

```
4  {
5      "Effect": "Allow",
6      "Action": [
7          "iot:AddThingToThingGroup",
8          "iot:AttachPolicy",
9          "iot:AttachThingPrincipal",
10         "iot:CreateKeysAndCertificate",
11         "iot:CreatePolicy",
12         "iot:CreateRoleAlias",
13         "iot:CreateThing",
14         "iot:CreateThingGroup",
15         "iot:DescribeEndpoint",
16         "iot:DescribeRoleAlias",
17         "iot:DescribeThingGroup",
18         "iot:GetPolicy",
19         "iam:GetRole",
20         "iam:CreateRole",
21         "iam:PassRole",
22         "iam:CreatePolicy",
23         "iam:AttachRolePolicy",
24         "iam:GetPolicy",
25         "sts:GetCallerIdentity"
26     ],
27     "Resource": "*"
28 },
29 {
30     "Effect": "Allow",
31     "Action": [
32         "greengrass>CreateDeployment",
33         "greengrass:DeleteDeployment"
34     ]
35 }
```

On the right side, there's an 'Edit statement' panel with a 'Select a statement' dropdown and a 'Add new statement' button. At the bottom of the policy editor, there's another 'Add new statement' button. The top right of the interface has tabs for 'Visual', 'JSON', and 'Actions'.

Step 4: Install the Greengrass Core Software

Rerun the Installer and Check If Core Device Is Healthy

The screenshot shows the AWS IoT Greengrass Core devices interface. At the top, there is a breadcrumb navigation: AWS IoT > Greengrass > Core devices. Below the navigation, the title "Greengrass core devices" is displayed with an "Info" link. A search bar labeled "Search by core device name" is present. On the right side of the header, there are three buttons: a white button with a "C" icon, a white button labeled "Configure cloud discovery", and an orange button labeled "Set up one core device". Below the header, there is a table with two columns: "Name" and "Status". The table contains one row with the name "CMPE583-GreengrassCore" and the status "Healthy". The status is highlighted with a green checkmark icon. To the right of the table, there is a pagination indicator showing "1" and a gear icon for settings. The bottom right corner of the table area has a small "▼" icon.

| Name | Status |
|--|---------------------------------------|
| CMPE583-GreengrassCore | Healthy 5 minutes ago |

Step 4: Install the Greengrass Core Software

The screenshot shows the AWS IAM Roles page. The role name is "GreengrassV2TokenExchangeRole". The role is described as "Role for Greengrass IoT things to interact with AWS services using token exchange service". There is a "Delete" button in the top right corner. Below the role name, there are tabs for "Permissions", "Trust relationships", "Tags", "Access Advisor", and "Revoke sessions". The "Permissions" tab is selected. Under "Permissions policies (1)", there is a single policy named "GreengrassV2TokenExchangeRoleAccess". The policy is listed as "Customer managed". Below the policy list, the policy document is displayed in JSON format:

```
1 [ {  
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                "s3:GetBucketLocation"  
12            ],  
13            "Resource": "*"  
14        }  
15    ]  
16 }
```

- A role is created when you installed the AWS IoT Greengrass Core software.
- If you did not specify a name, the default is role name is *GreengrassV2TokenExchangeRole*.
- This role should have a policy named *GreengrassV2TokenExchangeRoleAccess*.
- If you want to use different Role and Policy names, check Installer help command.

★ You can delete the user that has been created after verifying the installation is successful!

Step 4: Install the Greengrass Core software

Troubleshooting

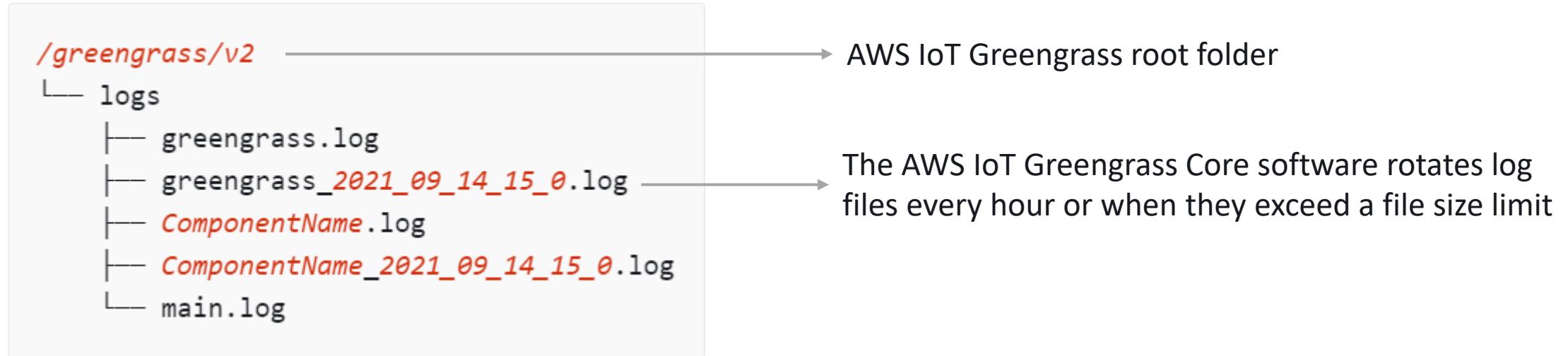
- If the GreengrassInstaller does not install the GreengrassV2TokenExchangeRole and prints an error message, you can create a custom role in IAM console with the following JSON value:

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

Step 4: Install the Greengrass Core software

Monitor AWS IoT Greengrass logs

- The AWS IoT Greengrass Core software stores logs in the /greengrass/v2/logs folder on a core device



- You must have root permissions to read AWS IoT Greengrass logs on the file system.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Step 4: Install the Greengrass Core Software

Auto Start of Greengrass Core Software

- If you installed the software as a system service, the installer runs the software at each startup.
- If you don't see following line in the installer output, Greengrass software will not start after rebooting the core device:

```
Successfully set up Nucleus as a system service
```

- In this case, you should run the software with the command below:

```
sudo /greengrass/v2/alts/current/distro/bin/loader
```

Uninstall the AWS IoT Greengrass Core Software For Linux Based Devices

- If the software runs as a system service, you must stop, disable, and remove it.

```
sudo systemctl stop greengrass.service  
sudo systemctl disable greengrass.service  
sudo rm /etc/systemd/system/greengrass.service
```

- Verify that the service is deleted.

```
sudo systemctl daemon-reload && sudo systemctl reset-failed
```

- Remove the root folder from the device.

```
sudo rm -rf /greengrass/v2
```

Greengrass Components

Greengrass Components

- A component is a software module that runs on AWS IoT Greengrass core devices.
- Every component is composed of a recipe and artifacts.

Recipe File

- Every component contains a recipe file.
- Recipes can be defined in JSON or YAML format.
- It defines component's metadata.
 - Configuration parameters
 - Component dependencies
 - Lifecycle
 - Platform compatibility

Artifacts

- Artifacts are component binaries.
- Components can have any number of artifacts.
- Artifacts can include:
 - Scripts
 - Compiled code
 - Static resources
 - Other files that a component consumes

Step 1: Upload the Artifacts

Create a bucket via AWS S3 (Console)

Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

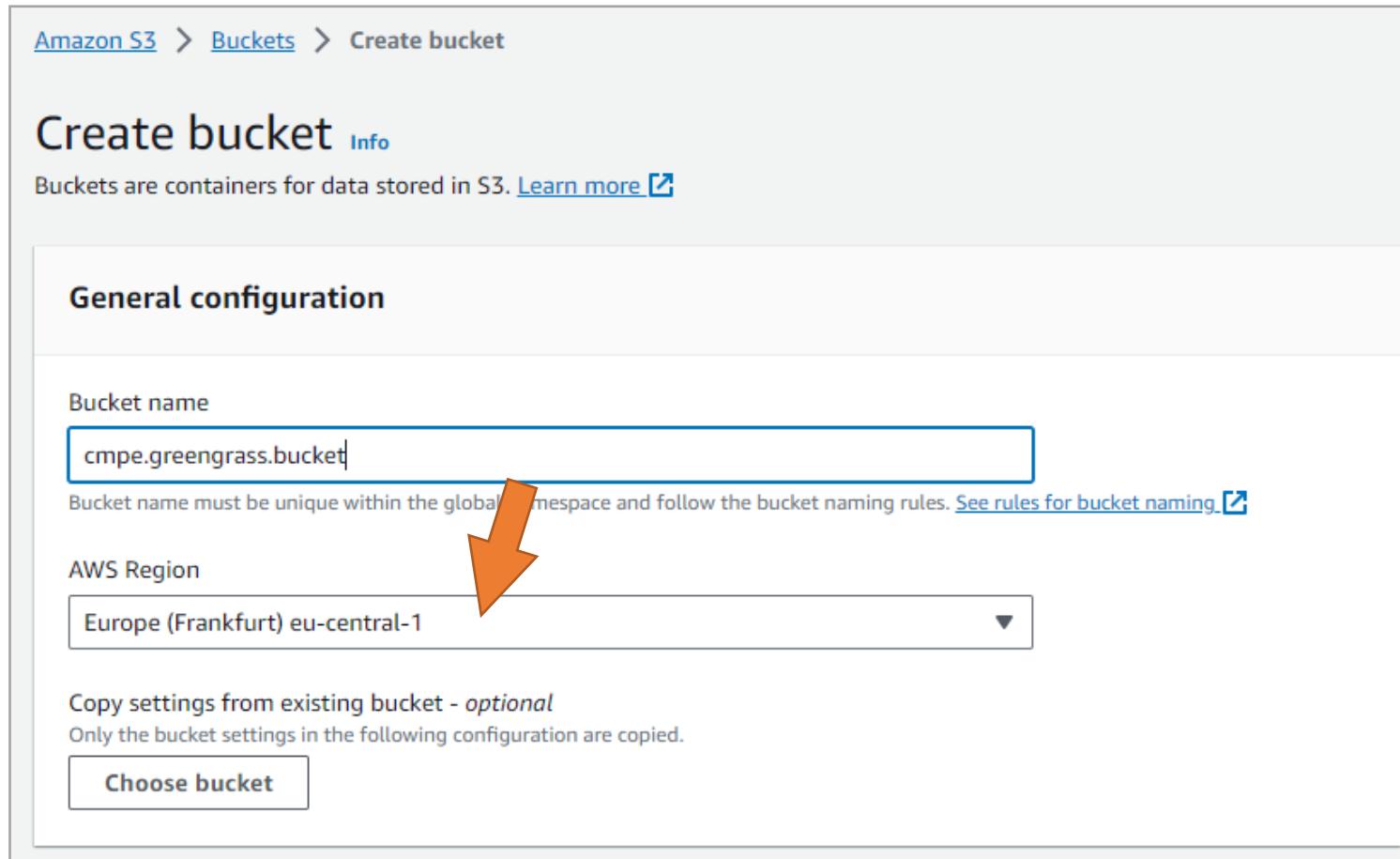
Bucket name

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)

AWS Region

Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

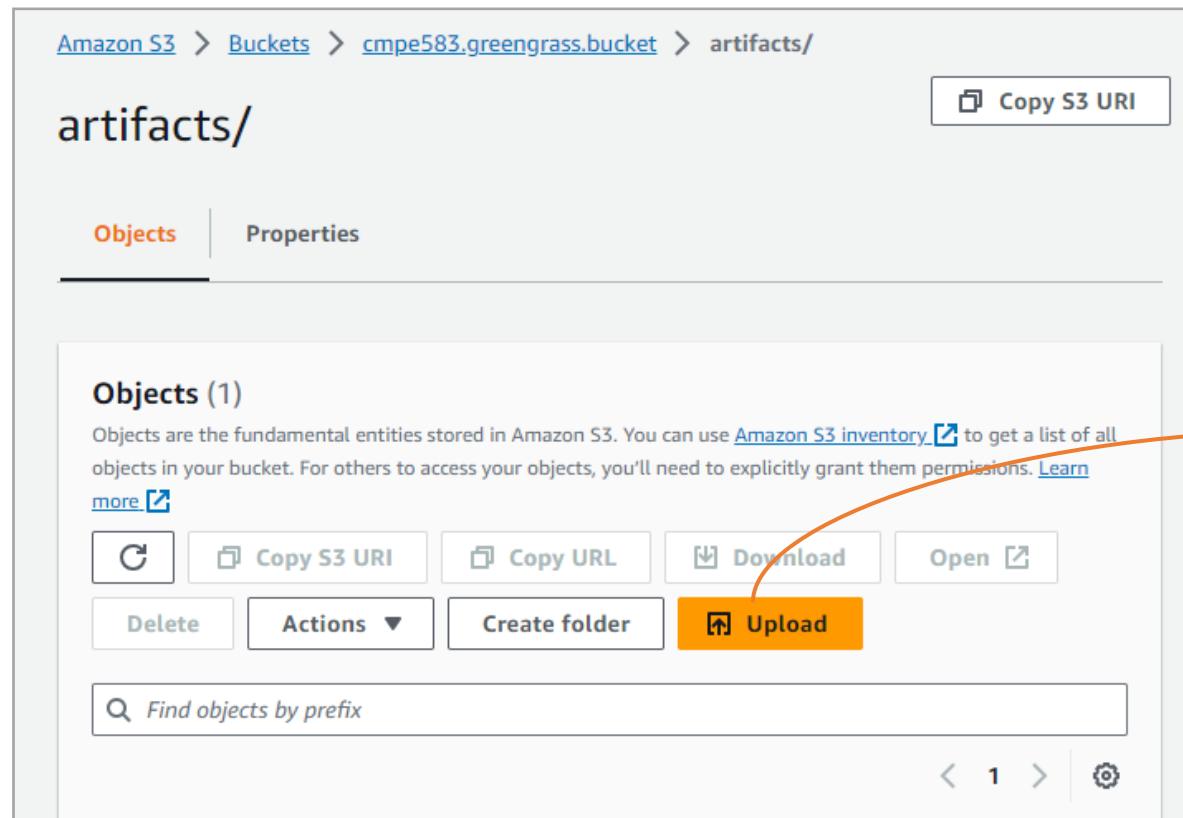


- Be sure that the S3 bucket is in the same AWS Region where you create the component.
- AWS IoT Greengrass doesn't support cross-Region requests for component artifacts!

Step 2: Upload the Artifacts

Create artifacts folder & upload files

- Upload your artifacts (source codes) to a folder in your S3 bucket.



```
cmpe583-PrintMsg.py
1 import sys
2
3 message = "Hi There, your message is: %s!" % sys.argv[1]
4
5 print(message)
```

Step 2: Set Policies to Access Files in S3

Via AWS IAM (Console)

- Greengrass component artifacts should be stored in AWS S3 bucket.
- Therefore, you should allow the core device to access component artifacts in the S3 bucket.
 - Attach a policy similar to policy below to GreengrassV2TokenExchangeRole from AWS IAM console:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [ "s3:GetObject" ],  
      "Resource": "arn:aws:s3:::Bucket-Name/*"  
    }  
  ]  
}
```

Step 2: Set Policies to Access Files in S3 Via AWS IAM (Console)

- You can create a new policy or simply attach an inline policy to related role in order to allow core device to Access objects in the S3 bucket.

IAM > Roles > GreengrassV2TokenExchangeRole

GreengrassV2TokenExchangeRole [Info](#)

Delete

Role for Greengrass IoT things to interact with AWS services using token exchange service

Permissions Trust relationships Tags Access Advisor Revoke sessions

Permissions policies (1) [Info](#)

You can attach up to 10 managed policies.

[C](#) [Simulate](#) [Remove](#) [Add permissions](#) ▲

[Attach policies](#)

[Create inline policy](#)

Filter by Type

Search

All types ▾

< 1 >

| <input type="checkbox"/> | Policy name Edit | Type | Attached entities |
|--------------------------|--|------------------|-------------------|
| <input type="checkbox"/> | GreengrassV2TokenExch... | Customer managed | 1 |

Specify permissions Info

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the Policy editor.

Policy editor

VisualJSON

```
1 ▼ {  
2     "Version": "2012-10-17",  
3     "Statement": [  
4         {  
5             "Effect": "Allow",  
6             "Action": ["s3:GetObject"],  
7             "Resource": "arn:aws:s3:::cmpe583.greengrass.bucket/*"  
8         }  
9     ]  
10 }
```

Edit statement

Select an existing statement
Add new statement

Step 3: Create the Component

Via AWS IoT Greengrass (Console)

The screenshot shows the 'Greengrass components' page in the AWS IoT Greengrass console. The navigation bar at the top shows 'AWS IoT > Greengrass > Components'. Below the navigation, there are three tabs: 'My components' (which is selected), 'Public components', and 'Community components'. A large orange arrow points to the 'Create component' button, which is located in the top right corner of the main content area. The main content area displays a message 'My components (0)' and a note: 'Your components are private components that only you can see and deploy to core devices.' Below this is a search bar with the placeholder 'Find by name, operating system, or architecture'. At the bottom of the page, there is a table header with columns: Name, Publisher, Version, Operating systems, Architectures, and Version created. The main message at the bottom states 'No components' and 'You don't have any Greengrass components in us-east-1.'. A final 'Create component' button is located at the bottom center.

AWS IoT > Greengrass > Components

Greengrass components Info

My components Public components Community components

Create component

My components (0)

Your components are private components that only you can see and deploy to core devices. [Learn more](#)

Find by name, operating system, or architecture

< 1 > 1 item

| Name | Publisher | Version | Operating systems | Architectures | Version created |
|------|-----------|---------|-------------------|---------------|-----------------|
|------|-----------|---------|-------------------|---------------|-----------------|

No components

You don't have any Greengrass components in us-east-1.

Create component

Step 3: Create the Component

Define the Recipe

AWS IoT > Greengrass > Components > Create component

Create component

When you finish your component, you can add it to AWS IoT Greengrass to deploy to core devices. Provide the component recipe and artifacts to create the component. This component is private and visible only to your AWS account.

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](#)

 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 ▼ {  
2     "RecipeFormatVersion": "2020-01-25",  
3     "ComponentName": "samplePrintMsg",  
4     "ComponentVersion": "1.0.0",  
5     "ComponentDescription": "My test AWS IoT Greengrass component.",  
6     "ComponentPublisher": "Amazon",  
7     "ComponentConfiguration": {  
8         "DefaultConfiguration": {"Message": "custom test message"}  
9     },  
10    "Manifests": [  
11        {  
12            "Platform": { "os": "linux" },  
13            "ManifestType": "aws-iot-device-sdk"  
14        }  
15    ]  
16}
```

Step 3: Create the Component

Configure the Recipe

```
{  
    "RecipeFormatVersion": "2020-01-25",  
    "ComponentName": "samplePrintMsg",  
    "ComponentVersion": "1.0.0",  
    "ComponentDescription": "My test AWS IoT Greengrass component.",  
    "ComponentPublisher": "Amazon",  
    "ComponentConfiguration": {  
        "DefaultConfiguration": {"Message": "custom test message"}  
    },  
    "Manifests": [  
        {  
            "Platform": { "os": "linux" },  
            "Lifecycle": { "run": "python3 -u {artifacts:path}/print_msg.py \\"{configuration:/Message}\\" " },  
            "Artifacts": [ { "URI": "s3://cmpe.greengrass.bucket/artifacts/print_msg.py" } ]  
        }  
    ]  
}
```

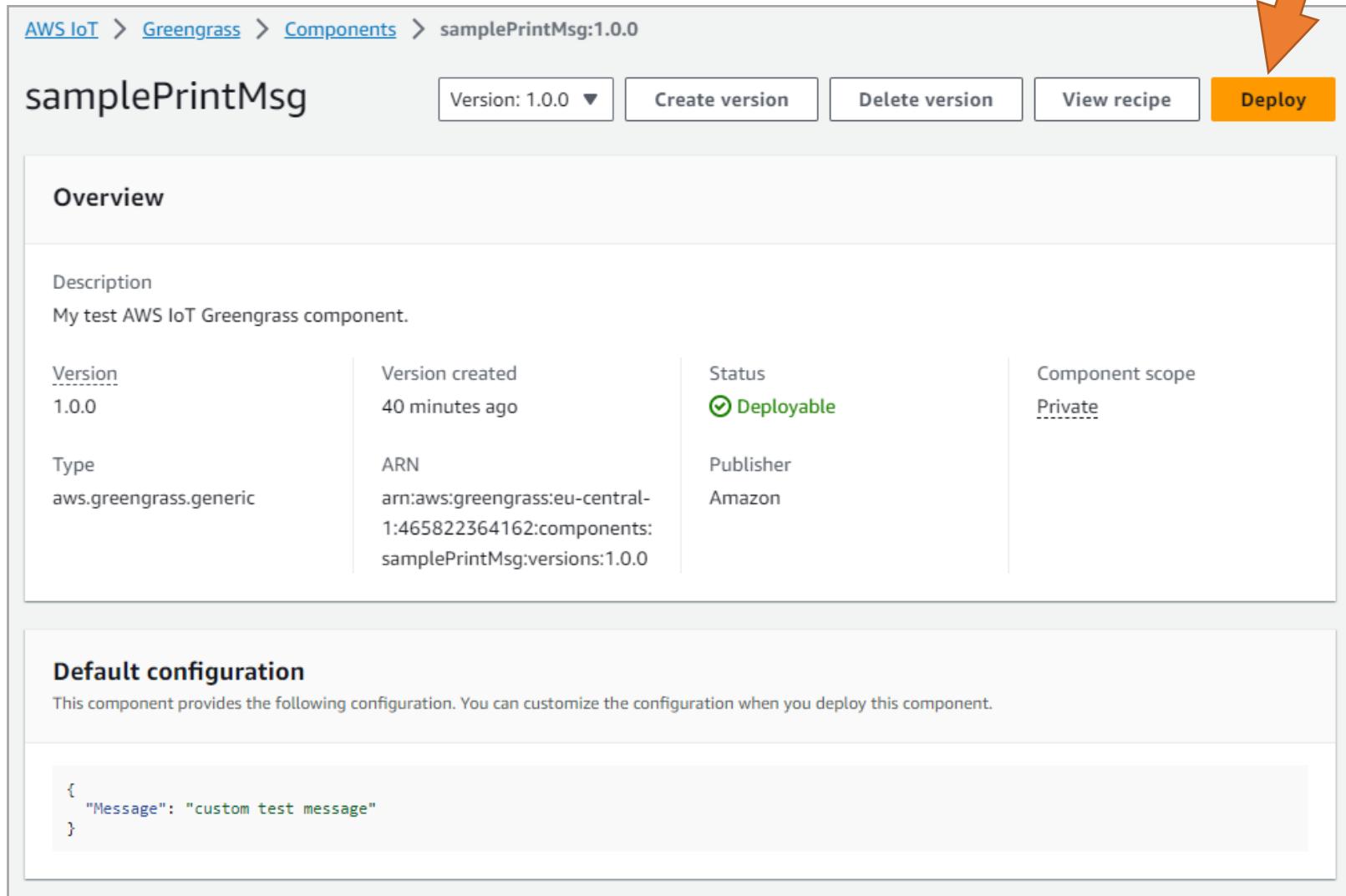
This test will be run on Linux device. So other OSs are ignored!

Define how to execute your artifact

Provide the artifact URI

Step 4: Deploy the Component

Via AWS IoT Greengrass (console)



AWS IoT > Greengrass > Components > samplePrintMsg:1.0.0

samplePrintMsg Version: 1.0.0 ▾ Create version Delete version View recipe Deploy

Overview

Description
My test AWS IoT Greengrass component.

| Version | Version created | Status | Component scope |
|------------------------|---|--|-----------------|
| 1.0.0 | 40 minutes ago | <input checked="" type="checkbox"/> Deployable | Private |
| Type | ARN | Publisher | |
| aws.greengrass.generic | arn:aws:greengrass:eu-central-1:465822364162:components:samplePrintMsg:versions:1.0.0 | Amazon | |

Default configuration

This component provides the following configuration. You can customize the configuration when you deploy this component.

```
{  
  "Message": "custom test message"  
}
```

Step 4: Deploy the Component

Configure Deployment

AWS IoT > Greengrass > Deployments > Create deployment

Step 1
Specify target

Step 2
Select components

Step 3 - optional
Configure components

Step 4 - optional
Configure advanced settings

Step 5
Review

Specify target

Deployment information

Name - *optional*
A friendly name lets you identify this deployment. If you leave it blank, the deployment displays its ID instead of a name.

Test Deployment for CMPE583

The deployment name can have up to 255 characters.

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

CMPE583-GreengrassGroup



al / Non-Personal Data

Step 4: Deploy the Component

Select the Component

AWS IoT > Greengrass > Deployments > Create deployment

Step 1
[Specify target](#)

Step 2
Select components

Step 3 - optional
[Configure components](#)

Step 4 - optional
[Configure advanced settings](#)

Step 5
[Review](#)

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 (1)

| <input checked="" type="checkbox"/> | Name  |
|-------------------------------------|--|
| <input checked="" type="checkbox"/> | samplePrintMsg |



Public components (47)

| <input type="checkbox"/> | Name  |
|--------------------------|--|
| <input type="checkbox"/> | aws.greengrass.SNS |
| <input type="checkbox"/> | aws.greengrass.LegacySubscriptionRouter |
| <input type="checkbox"/> | aws.greengrass.clientdevices.mqtt.Bridge |

Step 4: Deploy the Component

Configure Advanced Settings

AWS IoT > Greengrass > Deployments > Create deployment

Step 1
[Specify target](#)

Step 2
[Select components](#)

Step 3 - optional
[Configure components](#)

Step 4 - optional
[Configure advanced settings](#)

Step 5
Review

Configure advanced settings - *optional*

You can configure advanced settings such as how much time each device has to apply the deployment.

▶ **Rollout configuration**
The rollout configuration defines the rate at which the configuration deploys to the target devices.

▶ **Timeout configuration**
The timeout configuration defines the duration that each device has to apply the deployment.

▶ **Cancel configuration**
The cancel configuration defines when to automatically stop the deployment. The deployment cancels if a percentage of devices fail the deployment after a minimum number deploy. The deployment cancels if any criteria is met during the deployment.

▶ **Deployment policies**
Deployment policies define how a deployment handles failure and updates to components that are running on the target devices.



Cancel Previous **Next**

Step 4: Deploy the Component

Review & Deploy

AWS IoT > Greengrass > Deployments > Create deployment

Step 1
[Specify target](#)

Step 2
[Select components](#)

Step 3 - optional
[Configure components](#)

Step 4 - optional
[Configure advanced settings](#)

Step 5
Review

Review

Step 4: Advanced deployment configurations

Advanced deployment configurations

Rollout configuration
Constant rate, Maximum number of devices per minute: 1000

Timeout configuration
This deployment doesn't specify a timeout configuration.

Cancel configuration
This deployment doesn't specify a cancel configuration.

Component update policy
Notify components, Component update response timeout: 60 second

Configuration validation response timeout
This deployment doesn't specify a configuration validation response timeout.

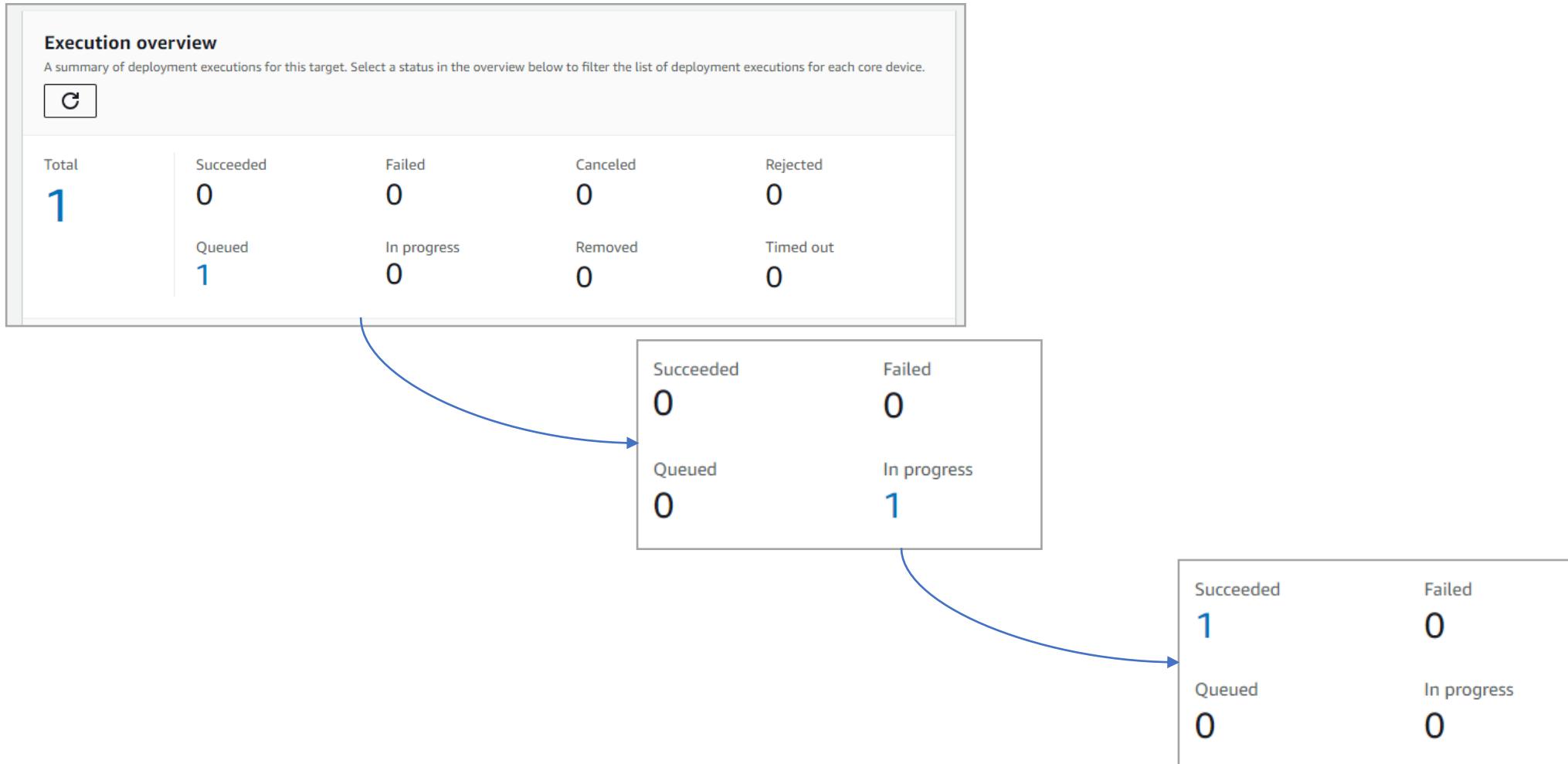
Failure handling policy
Rollback

 **Deploy**

[Download as JSON](#) [Cancel](#) [Previous](#) **Deploy**

Step 5: Verify the Component

Check AWS IoT Greengrass Console



Step 5: Verify the Component

Check Log Files

- Check the output log file created by your component to verify everything goes well.

```
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo# ls /greengrass/v2/logs/
aws.greengrass.Nucleus.log  greengrass_2023_10_29_15_0.log  greengrass.log  main.log  samplePrintMsg.log
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo# tail -n 100 -f /greengrass/v2/logs/samplePrintMsg.log
2023-10-29T13:04:00.263Z [INFO] (pool-2-thread-18) samplePrintMsg: shell-runner-start. {scriptName=services.samplePrintMsg.lifecycle.run, serviceName=samplePrintMsg, currentState=STARTING, command=["python3 -u /greengrass/v2/packages/artifacts/samplePrintMsg/1.0.0/cmpe583-Print..."]}
2023-10-29T13:04:00.305Z [INFO] (Copier) samplePrintMsg: stdout. Hi There, your message is: custom test message!. {scriptName=services.samplePrintMsg.lifecycle.run, serviceName=samplePrintMsg, currentState=RUNNING}
2023-10-29T13:04:00.310Z [INFO] (Copier) samplePrintMsg: Run script exited. {exitCode=0, serviceName=samplePrintMsg, currentState=RUNNING}
```



All Good!

Troubleshooting

AWS IoT > Greengrass > Deployments > Test Deployment for CMPE583

Test Deployment for CMPE583

Latest revision: 1 ▾ Cancel Actions ▾

This deployment targets an AWS IoT thing group. Add a core device to the thing group to apply this deployment to it.

Overview

AWS IoT Greengrass uses continuous AWS IoT jobs to deploy to thing groups.

| | | |
|---|--|------------------------------------|
| Target CMPE583-GreengrassGroup | Target type Thing group | Deployment created 1 minute ago |
| IoT job 3ae1680c-7f4c-4068-86ea-88f9e4a4e038 | Deployment status Active | |

Executions Devices Components Configurations Subdeployments Tags

Execution overview

A summary of deployment executions for this target. Select a status in the overview below to filter the list of deployment executions for each core device.

| | | | | |
|-------------------|------------------|----------------------|----------------|---------------|
| Total 1 | Succeeded 0 | Failed 1 ← | Cancelled 0 | Rejected 0 |
| Queued 0 | In progress 0 | Removed 0 | Timed out 0 | |

Troubleshooting

Check greengrass.log File

- Be sure that your IAM role is sufficient to access any resources defined in the deployment!

```
2023-09-23T10:40:57.822Z [INFO] (pool-2-thread-27) com.aws.greengrass.tes.CredentialRequestHandler: Received credentials that will be cached until 2023-09-23T11:35:57Z. {iotCredentialsPath=/role-aliases/GreengrassV2TokenExchangeRoleAlias/credentials}
2023-09-23T10:40:58.441Z [ERROR] (pool-2-thread-27) com.aws.greengrass.componentmanager.ComponentManager: Failed to prepare package com.boun.cmpe583-v1.0.0. {}
com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact name: 's3://greengrass.test.bucket/artifacts/print_msg.py' for component com.boun.cmpe583-1.0.0, reason: S3 HeadObject returns 403 Access Denied. Ensure the IAM role associated with the core device has a policy granting s3:GetObject
    at com.aws.greengrass.componentmanager.builtins.S3Downloader.getDownloadSize(S3Downloader.java:171)
    at com.aws.greengrass.componentmanager.ComponentManager.prepareArtifacts(ComponentManager.java:441)
    at com.aws.greengrass.componentmanager.ComponentManager.preparePackage(ComponentManager.java:397)
    at com.aws.greengrass.componentmanager.ComponentManager.lambda$preparePackages$1(ComponentManager.java:358)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by: software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: BVCVFQ32XV9H149X, Extended Request ID: jlg8Vcxx9IKjUwa/zzbgfRR/ujdTbrA9ZUoYzvrvqSQY7z8LN4vjtzdD+yD+ZqTXs0dUqHV65uA=)
    at software.amazon.awssdk.protocols.xml.internal.unmarshall.AwsXmlPredicatedResponseHandler.handleErrorResponse(AwsXmlPredicatedResponseHandler.java:156)
    at software.amazon.awssdk.protocols.xml.internal.unmarshall.AwsXmlPredicatedResponseHandler.handleResponse(AwsXmlPredicatedResponseHandler.java:108)
    at software.amazon.awssdk.protocols.xml.internal.unmarshall.AwsXmlPredicatedResponseHandler.handle(AwsXmlPredicatedResponseHandler.java:85)
    at software.amazon.awssdk.protocols.xml.internal.unmarshall.AwsXmlPredicatedResponseHandler.handle(AwsXmlPredicatedResponseHandler.java:43)
    at software.amazon.awssdk.awscore.client.handler AwsSyncClientHandler$Crc32ValidationResponseHandler.handle(AwsSyncClientHandler.java:95)
    at software.amazon.awssdk.core.internal.handler.BaseClientHandler.lambda$successTransformationResponseHandler$7(BaseClientHandler.java:264)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.HandleResponseStage.execute(HandleResponseStage.java:40)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.HandleResponseStage.execute(HandleResponseStage.java:30)
    at software.amazon.awssdk.core.internal.http.pipeline.RequestPipelineBuilder$ComposingRequestPipelineStage.execute(RequestPipelineBuilder.java:206)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptTimeoutTrackingStage.execute(ApiCallAttemptTimeoutTrackingStage.java:73)
    at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptTimeoutTrackingStage.execute(ApiCallAttemptTimeoutTrackingStage.java:42)
```



AWS Provided Components

AWS Provided Components

Nucleus

- AWS IoT Greengrass provides and maintains prebuilt components that you can deploy to your devices.
- Several AWS-provided components depend on specific minor versions of the nucleus.
- The nucleus is a mandatory component and the minimum requirement to run the AWS IoT Greengrass Core software on a device.
- It manages deployments, orchestration, and lifecycle management of other components.
- You need to update these components when you update the Greengrass nucleus to a new minor version.

AWS Provided Components

OS Support and More



| Component | Description | Depends on nucleus | Component type | Supported OS | Open source |
|--|---|--------------------|-----------------|----------------|-------------|
| Greengrass nucleus | The nucleus of the AWS IoT Greengrass Core software. Use this component to configure and update the software on your core devices. | - | Nucleus | Linux, Windows | Yes |
| Client device auth | Enables local IoT devices, called client devices, to connect to the core device. | Yes | Plugin | Linux, Windows | Yes |
| CloudWatch metrics | Publishes custom metrics to Amazon CloudWatch. | Yes | Generic, Lambda | Linux, Windows | Yes |
| AWS IoT Device Defender | Notifies administrators of changes in the state of the Greengrass core device to identify unusual behavior. | Yes | Generic, Lambda | Linux, Windows | Yes |
| Disk spooler | Enables a persistent storage option for messages spooled from Greengrass core devices to AWS IoT Core. This component will store these outbound messages on disk. | Yes | Plugin | Linux, Windows | Yes |
| Docker application manager | Enables AWS IoT Greengrass to download Docker images from Docker Hub and Amazon Elastic Container Registry (Amazon ECR). | Yes | Generic | Linux, Windows | No |
| Edge connector for Kinesis Video Streams | Reads video feeds from local cameras, publishes the streams to Kinesis Video Streams, and displays the streams in Grafana dashboards with AWS IoT TwinMaker. | Yes | Generic | Linux | No |
| Greengrass CLI | Provides a command-line interface that you can use to create local deployments and interact with the Greengrass core device and its components. | Yes | Plugin | Linux, Windows | Yes |
| IP detector | Reports MQTT broker connectivity information to AWS IoT Greengrass, so client devices can discover how to connect. | Yes | Plugin | Linux, Windows | Yes |
| Kinesis Data Firehose | Publishes data through Amazon Kinesis Data Firehose delivery streams to destinations in the AWS Cloud. | Yes | Lambda | Linux | No |
| Lambda launcher | Handles processes and environment configuration for Lambda functions. | No | Generic | Linux | No |
| Lambda manager | Handles interprocess communication and scaling for Lambda functions. | Yes | Plugin | Linux | No |
| Lambda runtimes | Provides artifacts for each Lambda runtime. | No | Generic | Linux | No |
| Legacy subscription router | Manages subscriptions for Lambda functions that run on AWS IoT Greengrass V1. | Yes | Generic | Linux | No |
| Local debug console | Provides a local console that you can use to debug and manage the Greengrass core device and its components. | Yes | Plugin | Linux, Windows | Yes |
| Log manager | Collects and uploads logs on the Greengrass core device. | Yes | Plugin | Linux, Windows | Yes |

Collecting IP Addresses of Core Devices

IPDetector

AWS IoT > Greengrass > Components

Greengrass components Info

My components | **Public components** | Community components

Public components (47)

AWS IoT Greengrass provides public components that you can deploy to core devices. You can deploy these components to use their standalone features, or you can use them as dependencies for your custom components. [Learn more](#) 

| Name | Publisher | Version | Architectures | Version created | Documentation |
|---|-----------|---------|---------------|-----------------|--|
| aws.greengrass.clientdevices.IPDetector | AWS | 2.1.7 | All | 3 months ago | Learn more  |

Search bar: ipde X 1 match < 1 > 



Collecting IP Addresses of Core Devices

Deploy an IPDetector Component

AWS IoT > Greengrass > Components > aws.greengrass.clientdevices.IPDetector:2.1.7

aws.greengrass.clientdevices.IPDetector

Version: 2.1.7 | View recipe | **Deploy**

Overview | View documentation

Description
The IP detector component reports the core device's connectivity information to the AWS IoT Greengrass cloud service. Client devices use this information to discover core devices to which they can connect.

| | | | |
|--------------------------------------|---|--|----------------------------------|
| Version 2.1.7 | Version created 3 months ago | Status ✓ Deployable | Component scope Public |
| Type aws.greengrass.plugin | ARN arn:aws:greengrass:eu-central-1:aws:components:aws.greengrass.clientdevices.IPDetector:versions:2.1.7 | Publisher AWS | |

Dependencies (1)

This component depends on these components. When you deploy this component, AWS IoT Greengrass also deploys a compatible dependency.

| Component | Version requirement | Dependency type |
|------------------------|---------------------|-----------------|
| aws.greengrass.Nucleus | >=2.2.0 <2.12.0 | Soft |

Collecting IP Addresses of Core Devices

Add Public Component to Existing Deployment

- You can add your component to an existing deployment, or create a new one!

Add to deployment

Deployment

Add to existing deployment Create new deployment

Find by deployment name or target name

| Deployment | Target name | Target type | Status | Deployment created |
|---|-------------------------|-------------|--------|--------------------|
| Test Deployment for CMPE583 | CMPE583-GreengrassGroup | Thing group | Active | 2 hours ago |

[Cancel](#) [Next](#)

Collecting IP Addresses of Core Devices

Select All Components You Want to Deploy

AWS IoT > Greengrass > Deployments > [c055d366-e31c-4221-bf0b-3d30c03aae06](#) > Revise deployment

Step 1
[Specify target](#)

Step 2 - optional
[Select components](#)

Step 3 - optional
[Configure components](#)

Step 4 - optional
[Configure advanced settings](#)

Step 5
[Review](#)

Select components - *optional*

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 (1)

| Name |
|--|
| <input checked="" type="checkbox"/> samplePrintMsg |

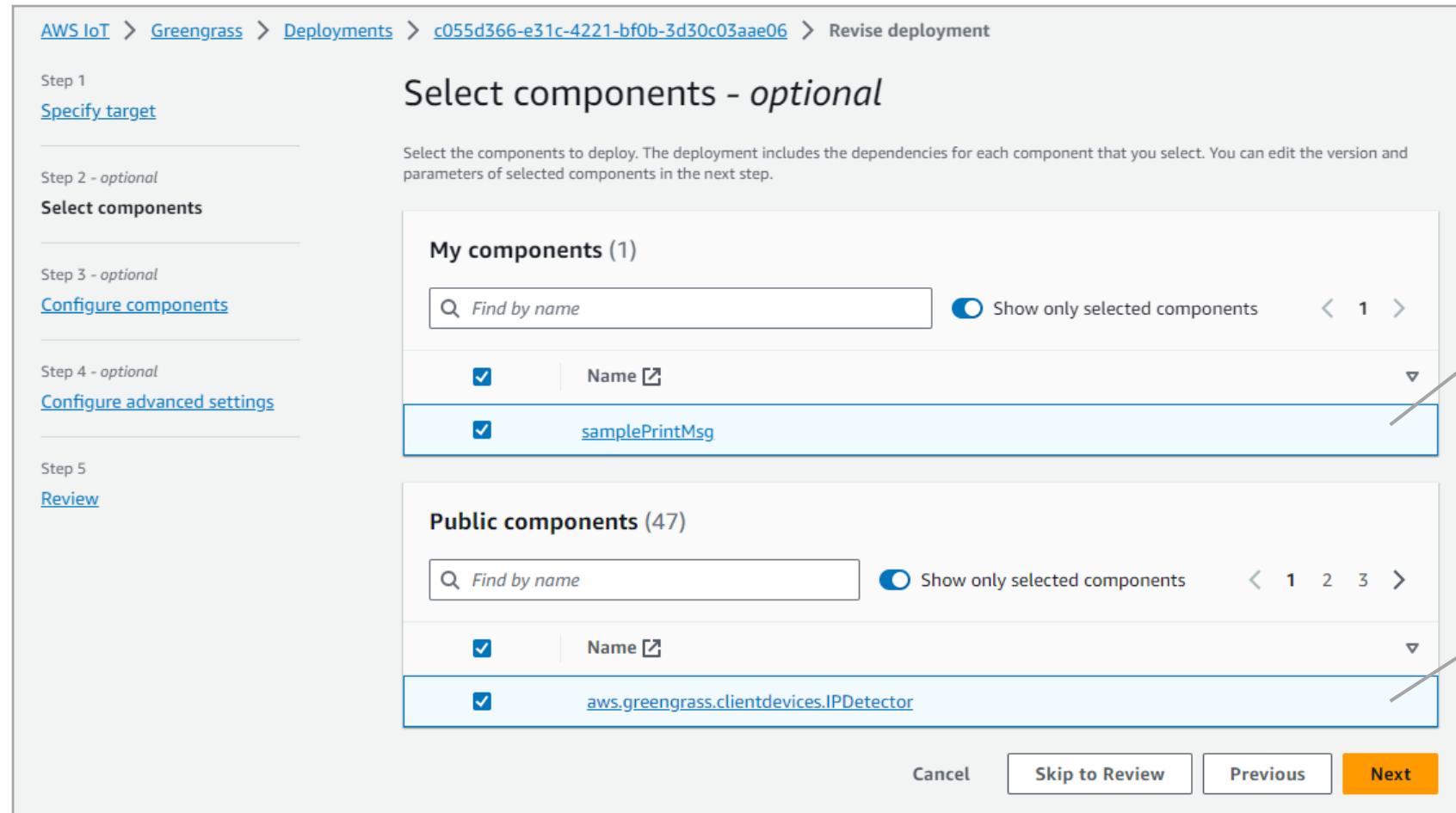
Public components (47)

| Name |
|---|
| <input checked="" type="checkbox"/> aws.greengrass.clientdevices.IPDetector |

Custom component

AWS Provided component

Cancel [Skip to Review](#) [Previous](#) **Next**



Collecting IP Addresses of Core Devices

Troubleshooting

- Check greengrass.log file if the deployed job runs without error.
- You will see an error if the Greengrass service role is not associated to your AWS account.
- Greengrass needs permission to access the AWS services on your behalf.

```
2023-09-23T13:03:40.754Z [WARN] (pool-1-thread-4) com.amazonaws.greengrass.detector.uploader.ConnectivityUpdater: Failed to upload the IP addresses. Make sure that the core device's IoT policy grants the greengrass:UpdateConnectivityInfo permission. Also the Greengrass service role must be associated to your AWS account with the iot:GetThingShadow and iot:UpdateThingShadow permissions.. {}  
software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: Could not find a Service Role associated with this account. (Service: Greengrass V2Data, Status Code: 403, Request ID: 0e740d21-2cf0-8ce6-27b4-a8ec8783f446)  
at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handleErrorResponse(CombinedResponseHandler.java:125)  
at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handleResponse(CombinedResponseHandler.java:82)  
at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handle(CombinedResponseHandler.java:60)  
at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handle(CombinedResponseHandler.java:41)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.HandleResponseStage.execute(HandleResponseStage.java:40)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.HandleResponseStage.execute(HandleResponseStage.java:30)  
at software.amazon.awssdk.core.internal.http.pipeline.RequestPipelineBuilder$ComposingRequestPipelineStage.execute(RequestPipelineBuilder.java:206)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptTimeoutTrackingStage.execute(ApiCallAttemptTimeoutTrackingStage.java:73)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptTimeoutTrackingStage.execute(ApiCallAttemptTimeoutTrackingStage.java:42)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.TimeoutExceptionHandlingStage.execute(TimeoutExceptionHandlingStage.java:78)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.TimeoutExceptionHandlingStage.execute(TimeoutExceptionHandlingStage.java:40)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptMetricCollectionStage.execute(ApiCallAttemptMetricCollectionStage.java:50)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptMetricCollectionStage.execute(ApiCallAttemptMetricCollectionStage.java:36)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.RetryableStage.execute(RetryableStage.java:81)  
at software.amazon.awssdk.core.internal.http.pipeline.stages.RetryableStage.execute(RetryableStage.java:36)  
at software.amazon.awssdk.core.internal.http.pipeline.RequestPipelineBuilder$ComposingRequestPipelineStage.execute(RequestPipelineBuilder.java:206)  
at software.amazon.awssdk.core.internal.http.StreamManagingStage.execute(StreamManagingStage.java:56)  
at software.amazon.awssdk.core.internal.http.StreamManagingStage.execute(StreamManagingStage.java:36)
```



Troubleshooting

Create an IAM Role

The screenshot shows the AWS Identity and Access Management (IAM) service interface. The left sidebar is titled "Identity and Access Management (IAM)" and includes sections for Dashboard, Access management (User groups, Users, Roles, Policies, Identity providers, Account settings), Access reports (Access analyzer, Archive rules, Analyzers, Settings), Credential report, and Organization activity. The "Roles" section is currently selected. The main content area is titled "Roles (20)" and contains a brief description: "An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust." Below this is a search bar and a "Create role" button, which is highlighted with a large orange arrow. A table lists 20 existing roles, each with a checkbox, a link to the role's details, and its trusted entity type. The first few rows are: "AWSServiceRoleForAccessAnalyzer" (AWS Service: accessanalyzer), "AWSServiceRoleForAmazonCodeGuruReviewer" (AWS Service: codeguru), "AWSServiceRoleForAmazonSSM" (AWS Service: ssm), "AWSServiceRoleForCloudFormationStackSetsOrgMember" (AWS Service: member), "AWSServiceRoleForCloudTrail" (AWS Service: cloudtrail), "AWSServiceRoleForElasticLoadBalancing" (AWS Service: elasticloadbalancing), "AWSServiceRoleForGlobalAccelerator" (AWS Service: globalaccelerator), "AWSServiceRoleForOrganizations" (AWS Service: organizations), "AWSServiceRoleForSSO" (AWS Service: sso), "AWSServiceRoleForSupport" (AWS Service: support), and "AWSServiceRoleForTrustedAdvisor" (AWS Service: trustedadvisor).

| Role name | Trusted entities |
|---|-----------------------------------|
| AWSServiceRoleForAccessAnalyzer | AWS Service: accessanalyzer |
| AWSServiceRoleForAmazonCodeGuruReviewer | AWS Service: codeguru |
| AWSServiceRoleForAmazonSSM | AWS Service: ssm |
| AWSServiceRoleForCloudFormationStackSetsOrgMember | AWS Service: member |
| AWSServiceRoleForCloudTrail | AWS Service: cloudtrail |
| AWSServiceRoleForElasticLoadBalancing | AWS Service: elasticloadbalancing |
| AWSServiceRoleForGlobalAccelerator | AWS Service: globalaccelerator |
| AWSServiceRoleForOrganizations | AWS Service: organizations |
| AWSServiceRoleForSSO | AWS Service: sso |
| AWSServiceRoleForSupport | AWS Service: support |
| AWSServiceRoleForTrustedAdvisor | AWS Service: trustedadvisor |

Troubleshooting

Create an IAM Role

IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity Info

Trusted entity type

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

AWS account
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

Web identity
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

Use case

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

Service or use case

Greengrass

Choose a use case for the specified service.

Use case

Greengrass
Allows Greengrass to call AWS services on your behalf.

- Create a role that allows an AWS service to perform actions on your behalf.
- Allows Greengrass to call AWS services on your behalf.

Troubleshooting

Add Permission to Created Role

- To allow AWS IoT Greengrass to access your resources, the Greengrass service role must be associated with your AWS account and specify AWS IoT Greengrass as a trusted entity.

Add permissions [Info](#)

Permissions policies (1/917) [Info](#)

Choose one or more policies to attach to your new role.

Filter by Type

Q green

| Policy name | Type | Description |
|---|-------------|---|
| <input type="checkbox"/> AWSGreengrassFullAccess | AWS managed | This policy gives full access to the AWS Greengrass configuration, management and deployment actions |
| <input type="checkbox"/> AWSGreengrassReadOnlyAccess | AWS managed | This policy gives read only access to the AWS Greengrass configuration, management and deployment actions |
| <input checked="" type="checkbox"/> AWSGreengrassResourceAccessRolePolicy | AWS managed | Policy for AWS Greengrass service role which allows access to related services including AWS Lambda and AWS IoT thing shadows. |
| <input type="checkbox"/> AWSIoTDeviceTesterForGreengrassFullAcc... | AWS managed | Allows AWS IoT Device Tester to run the AWS Greengrass qualification suite by allowing access to related services including Lambda, ... |
| <input type="checkbox"/> AWSPanoramaGreengrassGroupRolePolicy | AWS managed | Allows an AWS Lambda function on an AWS Panorama Appliance to manage resources in Panorama, upload logs and metrics to Ama... |
| <input type="checkbox"/> GreengrassOTAUpdateArtifactAccess | AWS managed | Provides read access to the Greengrass OTA Update artifacts in all Greengrass regions |

▶ Set permissions boundary - *optional*

[Cancel](#) [Previous](#) [Next](#)

Troubleshooting

Review and Create Role

The screenshot shows the 'Create role' wizard in the AWS IAM console. The navigation path is 'IAM > Roles > Create role'. The current step is 'Step 1: Select trusted entity', which is completed. The next step, 'Step 2: Add permissions', is shown below it. The final step, 'Step 3: Name, review, and create', is the active screen.

Name, review, and create

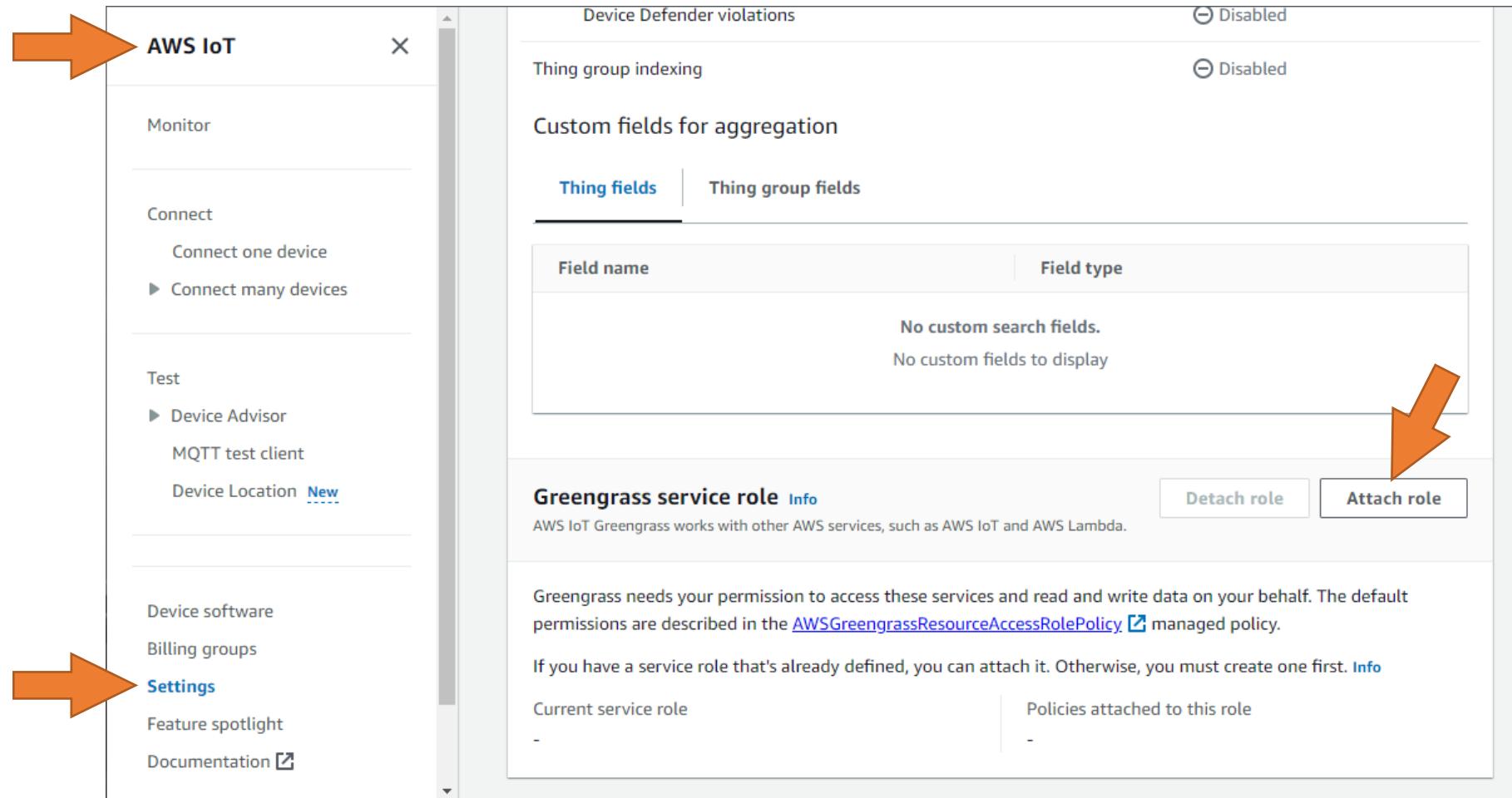
Role details

Role name
Enter a meaningful name to identify this role.
Maximum 64 characters. Use alphanumeric and '+,.@-_ ' characters.

Description
Add a short explanation for this role.
Maximum 1000 characters. Use alphanumeric and '+,.@-_ ' characters.

Troubleshooting

Attach Role to AWS IoT Service



Troubleshooting

Attach Role to AWS IoT Service

The screenshot shows the AWS IoT Greengrass service role configuration page. On the left, there's a sidebar with navigation links like Monitor, Connect, Test, Device software, Billing groups, Settings (which is selected), Feature spotlight, and Documentation. The main content area has tabs for Thing fields and Thing group fields, both currently disabled. Below this is a table with columns for Field name and Field type, showing 'No custom search fields.' and 'No custom fields to display'. A section titled 'Greengrass service role' includes a link to its info, a 'Detach role' button, and a 'Change role' button. It also contains a note about AWS IoT Greengrass working with other AWS services and a link to the AWSGreengrassResourceAccessRolePolicy managed policy. A success message box says 'Greengrass service role has been updated successfully.' At the bottom, it shows the current service role as 'arn:aws:iam::465822364162:role/CMPE583-GreengrassRole' and the attached policy as 'AWSGreengrassResourceAccessRolePolicy'.

AWS IoT

Device Defender violations

Thing fields Thing group fields

Field name Field type

No custom search fields.
No custom fields to display

Greengrass service role [Info](#)

AWS IoT Greengrass works with other AWS services, such as AWS IoT and AWS Lambda.

Detach role Change role

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](#)

Greengrass service role has been updated successfully.

Current service role
arn:aws:iam::465822364162:role/CMPE583-GreengrassRole

Policies attached to this role
AWSGreengrassResourceAccessRolePolicy

Troubleshooting

Verify the Deployment

- You can see the IP addresses of your core devices after the deployment succeeded.

The image shows two screenshots from the AWS IoT Greengrass console. The left screenshot is the 'Execution overview' page for a target, showing one deployment execution that has succeeded. An orange arrow points from the 'Succeeded' count on this page to the 'MQTT broker endpoints' section of the right screenshot. The right screenshot is the 'Core devices' page for a specific device named 'CMPE583-GreengrassCore'. It displays the 'Client devices' tab, which lists an MQTT broker endpoint with the endpoint address '10.0.2.15' and port '8883'.

AWS IoT > Greengrass > Core devices > CMPE583-GreengrassCore

CMPE583-GreengrassCore

Components Deployments Thing groups Client devices Tags

MQTT broker endpoints (1) [Info](#)

The endpoints where client devices can connect to an MQTT broker on this core device. If your client devices are on the same local network as this core device, you can deploy the [IP detector](#) component to manage the MQTT broker endpoints for you. Otherwise, you can manage the endpoints manually.

Manage endpoints

| Endpoint | Port | Connection metadata |
|-----------|------|---------------------|
| 10.0.2.15 | 8883 | |

Publish/Subscribe AWS IoT Core MQTT Messages

AWS IoT Greengrass Core IPC

- Components running on your core device can use the AWS IoT Greengrass Core interprocess communication (IPC) library in the AWS IoT Device SDK to communicate with the AWS IoT Greengrass nucleus and other Greengrass components.
- The IPC interface supports two types of operations:
 - **Request/response**
 - Components send a request to the IPC service and receive a response that contains the result of the request.
 - **Subscription**
 - Components send a subscription request to the IPC service and expect a stream of event messages in response.

AWS IoT Core MQTT Messaging IPC

- The AWS IoT Core MQTT messaging IPC service lets you send and receive MQTT messages to and from AWS IoT Core.
- Components can publish messages to AWS IoT Core and subscribe to topics to act on MQTT messages from other sources.
- To use AWS IoT Core MQTT messaging in a custom component, you must define authorization policies that allow your component to send and receive messages on topics.
 - **aws.greengrass#PublishToIoTCore**
 - Allows a component to publish messages to AWS IoT Core on the MQTT topics.
 - **aws.greengrass#SubscribeToIoTCore**
 - Allows a component to subscribe to messages from AWS IoT Core on the topics.

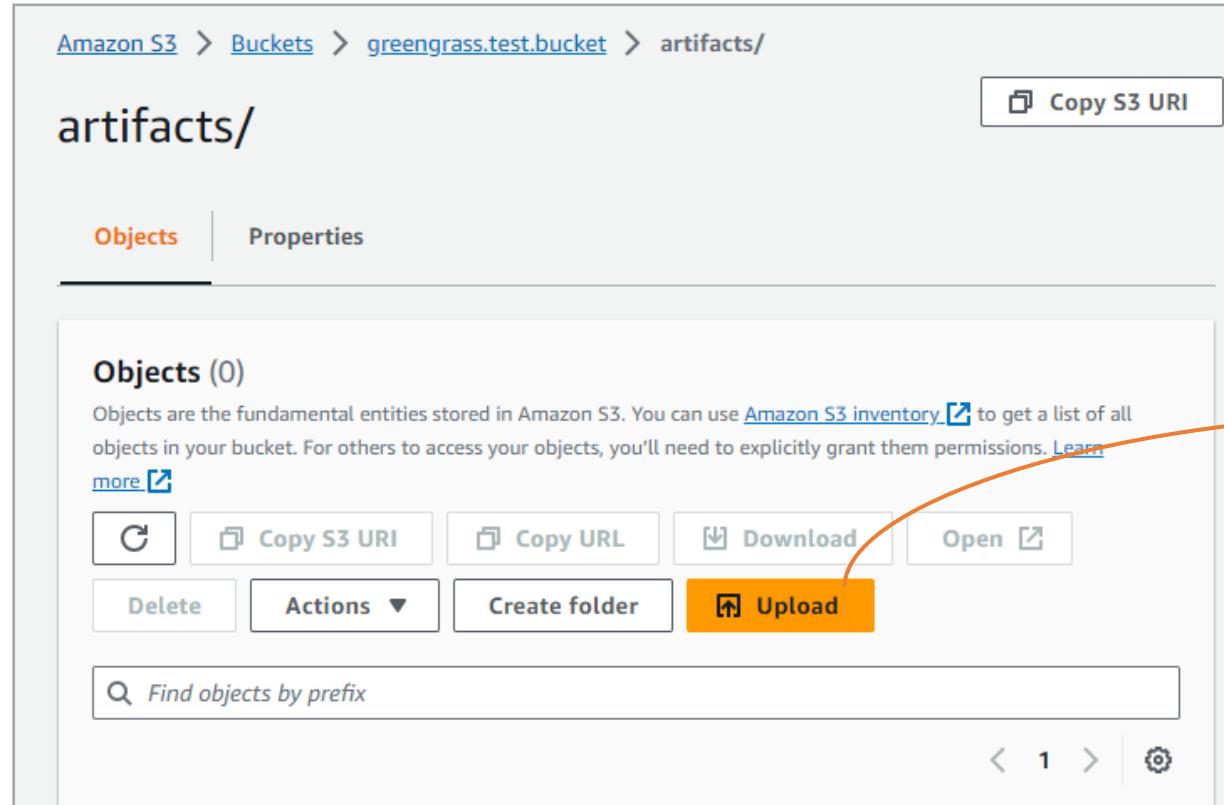
MQTT

- MQTT (Message Queuing Telemetry Transport) is a lightweight and widely adopted messaging protocol that is designed for constrained devices.
- AWS IoT Core support for MQTT is based on the MQTT v3.1.1 specification and the MQTT v5.0 specification.
- As the latest version of the standard, MQTT 5 introduces several key features that make an MQTT-based system more robust.
- AWS IoT Core also supports cross MQTT version (MQTT 3 and MQTT 5) communication.

Step 1: Develop & Upload Client-Side Code

Via AWS IoT Greengrass (Console)

- Upload your artifacts to a folder in your S3 bucket



```
cmpe583-PubSub.py x
from datetime import datetime
import time
import traceback
import json
import botocore
import sys
import os

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    IoTCoreMessage,
    QOS,
    SubscribeToIoTCoreRequest,
    PublishToIoTCoreRequest
)

TIMEOUT = 10
REQUEST_TOPIC = sys.argv[1]
RESPONSE_TOPIC = sys.argv[2]
THING_NAME = os.getenv('AWS_IOT_THING_NAME')

ipc_client = awsiot.greengrasscoreipc.connect()
...  
Sensitivity: Internal / Non-Personal Data
```

Step 2: Create a Custom Component

Via AWS IoT Greengrass (Console)

The screenshot shows the 'Greengrass components' page in the AWS IoT Greengrass console. The navigation bar at the top includes 'AWS IoT > Greengrass > Components'. Below the navigation, there are three tabs: 'My components' (which is selected), 'Public components', and 'Community components'. A large orange arrow points to the 'Create component' button, which is located in the top right corner of the main content area. The main content area displays a section titled 'My components (0)' with a note: 'Your components are private components that only you can see and deploy to core devices.' It includes a search bar labeled 'Find by name, operating system, or architecture' and a pagination indicator showing '1' of '1' pages. At the bottom, there is a table header with columns: Name, Publisher, Version, Operating systems, Architectures, and Version created. The main message in the center states 'No components' and 'You don't have any Greengrass components in us-east-1.'. A final 'Create component' button is located at the bottom center.

Step 2: Create a Custom Component

Enter the Recipe

AWS IoT > Greengrass > Components > Create component

Create component

When you finish your component, you can add it to AWS IoT Greengrass to deploy to core devices. Provide the component recipe and artifacts to create the component. This component is private and visible only to your AWS account.

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 ▼ {
2     "RecipeFormatVersion": "2020-01-25",
3     "ComponentName": "samplePubSub",
4     "ComponentVersion": "1.0.0",
5     "ComponentType": "aws.greengrass.generic",
6     "ComponentDescription": "A component that subscribes to a topic and responds back",
7     "ComponentPublisher": "<Name>",
8     "ComponentConfiguration": {
9         "DefaultConfiguration": {
10             "accessControl": {
11                 "aws.greengrass.ipc.mqtproxy": {
12                     "com.example.MyIoTCorePubSubComponent:mqtproxy:1": [
13                         "policyDescription": "Allows access to pub/sub to all topics.",
14                         "operations": [
15                             "aws.greengrass#PublishToIoTCore",
16                             "aws.greengrass#SubscribeToIoTCore"
17                         ]
18                     ]
19                 }
20             }
21         }
22     }
23 }
```

Step 2: Create a Custom Component

Adjust the Access Control

```
{  
  "accessControl": {  
    "aws.greengrass.ipc.mqttproxy": {  
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {  
        "policyDescription": "Allows access to publish/subscribe to all topics.",  
        "operations": [  
          "aws.greengrass#PublishToIoTCore",  
          "aws.greengrass#SubscribeToIoTCore"  
        ],  
        "resources": [ "*" ]  
      }  
    },  
    "ComponentDependencies": { ... },  
    "Manifests": [ ... ],  
    "Lifecycle": {}  
  }  
}
```

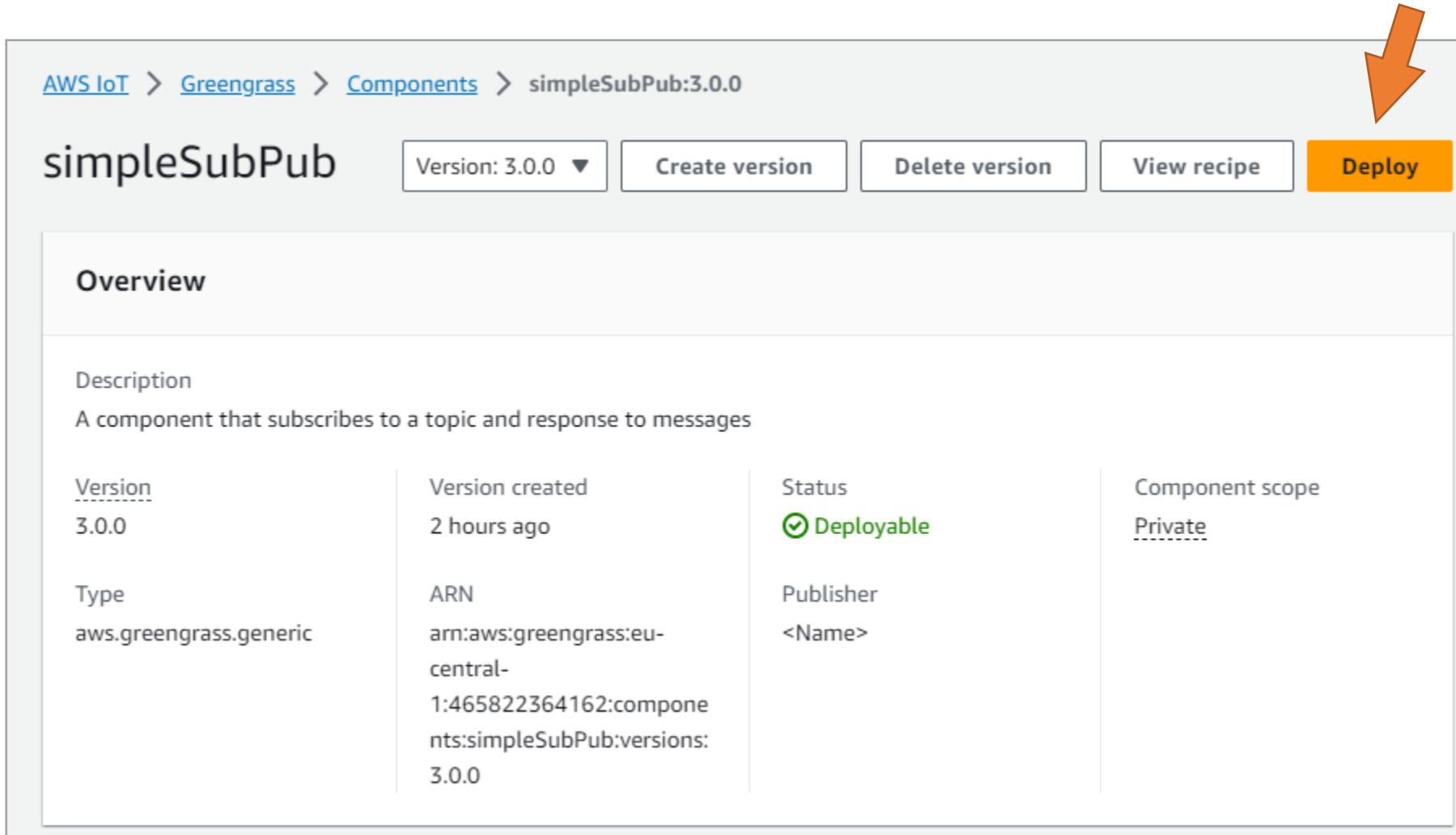
The diagram shows two callout boxes pointing from specific parts of the JSON code. One arrow points from the curly braces after 'aws.greengrass#SubscribeToIoTCore' to a box explaining the operations. Another arrow points from the curly braces after '*' to a box explaining the resources.

Allow the component to publish messages to AWS IoT Core or subscribe to messages from AWS IoT Core.

A topic string, such as test/topic, or * to allow access to all topics. You can use MQTT topic wildcards (# and +) to match multiple resources.

Step 3: Deploy Your Component

Via AWS IoT Greengrass (Console)



AWS IoT > Greengrass > Components > simpleSubPub:3.0.0

simpleSubPub Version: 3.0.0 ▾ Create version Delete version View recipe Deploy

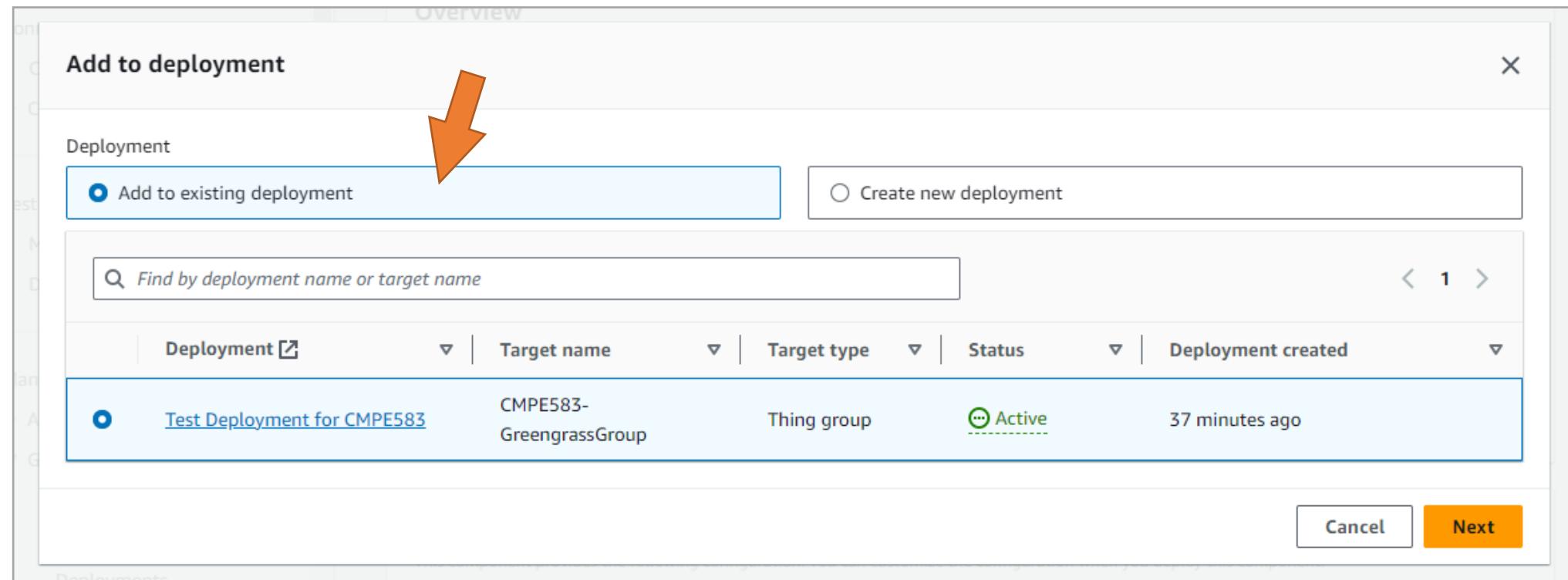
Overview

Description
A component that subscribes to a topic and response to messages

| Version | Version created | Status | Component scope |
|------------------------|---|--|-----------------|
| 3.0.0 | 2 hours ago | <input checked="" type="checkbox"/> Deployable | Private |
| Type | ARN | Publisher | |
| aws.greengrass.generic | arn:aws:greengrass:eu-central-1:465822364162:components:simpleSubPub:versions:3.0.0 | <Name> | |

Step 3: Deploy Your Component

Add Custom Component to Existing Deployment



Step 3: Deploy Your Component

Select the Component

Select components - *optional*

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 (2)

| <input checked="" type="checkbox"/> | Name  |
|-------------------------------------|---|
| <input checked="" type="checkbox"/> | samplePubSub |
| <input checked="" type="checkbox"/> | samplePrintMsg |

Public components (47)

| <input checked="" type="checkbox"/> | Name  |
|-------------------------------------|---|
| <input checked="" type="checkbox"/> | aws.greengrass.clientdevices.IPDetector |



Step 3: Deploy Your Component

Configure Advanced Settings

AWS IoT > Greengrass > Deployments > Create deployment

Step 1
[Specify target](#)

Step 2
[Select components](#)

Step 3 - optional
[Configure components](#)

Step 4 - optional
[Configure advanced settings](#)

Step 5
Review

Configure advanced settings - *optional*

You can configure advanced settings such as how much time each device has to apply the deployment.

▶ **Rollout configuration**
The rollout configuration defines the rate at which the configuration deploys to the target devices.

▶ **Timeout configuration**
The timeout configuration defines the duration that each device has to apply the deployment.

▶ **Cancel configuration**
The cancel configuration defines when to automatically stop the deployment. The deployment cancels if a percentage of devices fail the deployment after a minimum number deploy. The deployment cancels if any criteria is met during the deployment.

▶ **Deployment policies**
Deployment policies define how a deployment handles failure and updates to components that are running on the target devices.



Cancel Previous **Next**

Step 3: Deploy Your Component

Review & Deploy

AWS IoT > Greengrass > Deployments > Create deployment

Step 1
[Specify target](#)

Step 2
[Select components](#)

Step 3 - optional
[Configure components](#)

Step 4 - optional
[Configure advanced settings](#)

Step 5
Review

Review

Step 4: Advanced deployment configurations

[Edit](#)

Advanced deployment configurations

Rollout configuration
Constant rate, Maximum number of devices per minute: 1000

Timeout configuration
This deployment doesn't specify a timeout configuration.

Cancel configuration
This deployment doesn't specify a cancel configuration.

Component update policy
Notify components, Component update response timeout: 60 second

Configuration validation response timeout
This deployment doesn't specify a configuration validation response timeout.

Failure handling policy
Rollback

 Deploy

Download as JSON Cancel Previous Deploy

Step 4: Verify the Component

Check AWS IoT Greengrass Console



Troubleshooting

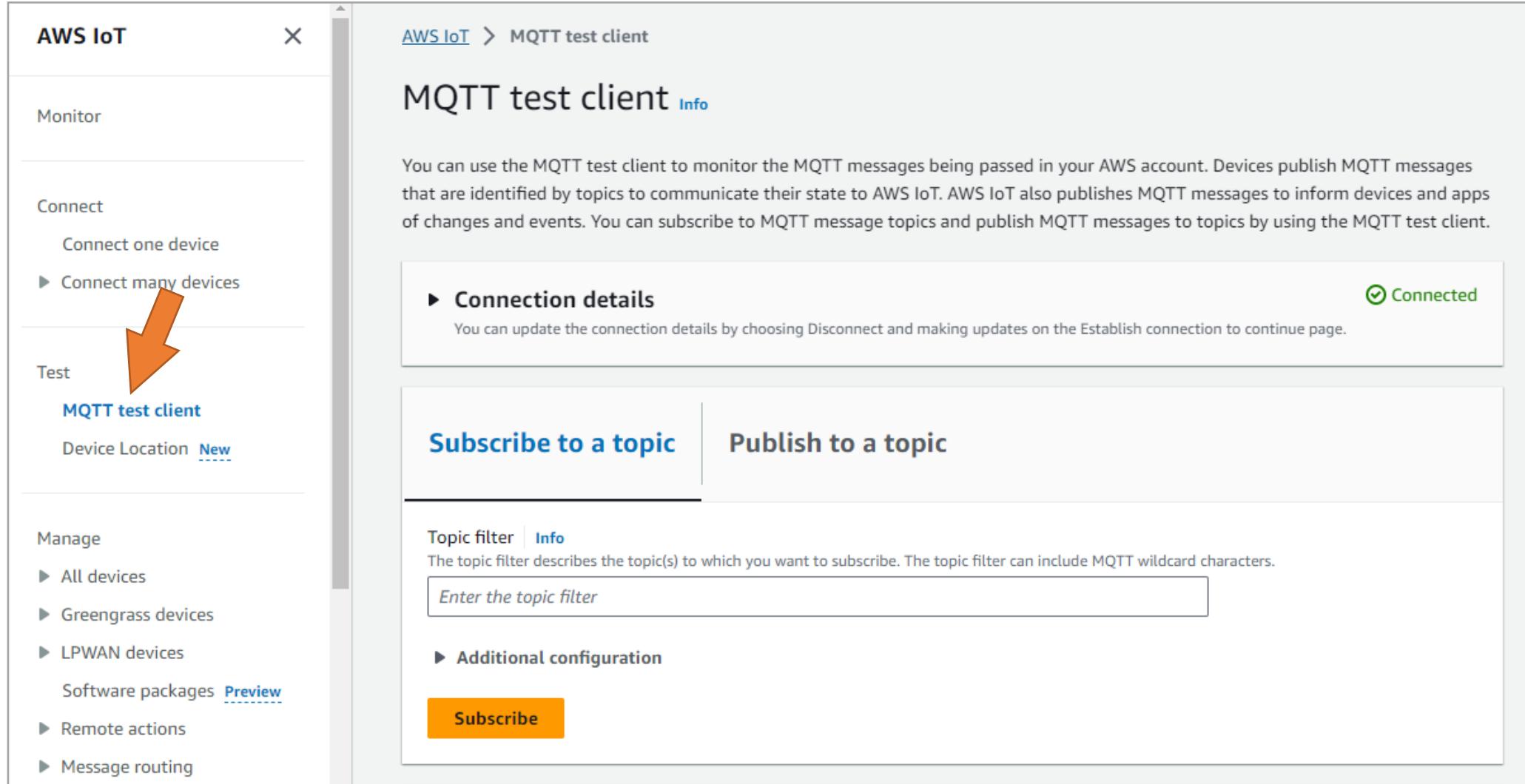
Check greengrass.log File

- Be sure that your Linux environment allows installing Python modules with pip utility!
- Modern Linux distributions mostly prevent installing modules system-wide.
- In this case use *pipx* or *--break-system-packages* option with pip3.

```
plePubSub.log
hread-32) samplePubSub: shell-runner-start. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW, command=["pip3 install xyz", {"scriptName": "services.samplePubSub.lifecycle.Install", "serviceName": "samplePubSub", "currentState": "NEW"}]
samplePubSub: stderr. error: externally-managed-environment. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. x This environment is externally managed. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. ↳ To install Python packages system-wide, try apt install. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. python3-xyz, where xyz is the package you are trying to. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. install.. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. If you wish to install a non-Debian-packaged Python package,. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. create a virtual environment using python3 -m venv path/to/venv.. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. sure you have python3-full installed.. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. If you wish to install a non-Debian packaged Python application,. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. it may be easiest to use pipx install xyz, which will manage a. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. virtual environment for you. Make sure you have pipx installed.. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. See /usr/share/doc/python3.12/README.venv for more information.. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
samplePubSub: stderr. {scriptName=services.samplePubSub.lifecycle.Install, serviceName=samplePubSub, currentState=NEW}
```

Test Your Deployment

Via AWS MQTT Test Client



The screenshot shows the AWS IoT console with the 'MQTT test client' page open. The left sidebar has a tree view with 'Monitor', 'Connect' (selected), 'Test' (selected), and 'Manage'. Under 'Test', 'MQTT test client' is highlighted and has an orange arrow pointing to it. The main content area shows the 'MQTT test client' page with a description of its purpose, connection details (Connected), and sections for 'Subscribe to a topic' and 'Publish to a topic'. The 'Topic filter' field is empty, and there is an 'Enter the topic filter' placeholder. The 'Subscribe' button is visible at the bottom of the 'Subscribe to a topic' section.

AWS IoT > MQTT test client

MQTT test client Info

You can use the MQTT test client to monitor the MQTT messages being passed in your AWS account. Devices publish MQTT messages that are identified by topics to communicate their state to AWS IoT. AWS IoT also publishes MQTT messages to inform devices and apps of changes and events. You can subscribe to MQTT message topics and publish MQTT messages to topics by using the MQTT test client.

Connection details Connected

You can update the connection details by choosing Disconnect and making updates on the Establish connection to continue page.

Subscribe to a topic **Publish to a topic**

Topic filter Info
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

Additional configuration

Subscribe

Test Your Deployment

Subscribe All Topics

The screenshot shows a user interface for testing MQTT subscriptions. At the top, there are two tabs: "Subscribe to a topic" (selected) and "Publish to a topic".

Subscribe to a topic:

- Topic filter:** A text input field containing the wildcard topic "#". An orange arrow points to this field.
- Info:** A small link next to the topic filter.
- Description:** A text explaining that the topic filter describes the topic(s) to which you want to subscribe, mentioning MQTT wildcard characters.
- Additional configuration:** A link to further configuration options.
- Subscribe button:** An orange button at the bottom of the form.

Subscriptions:

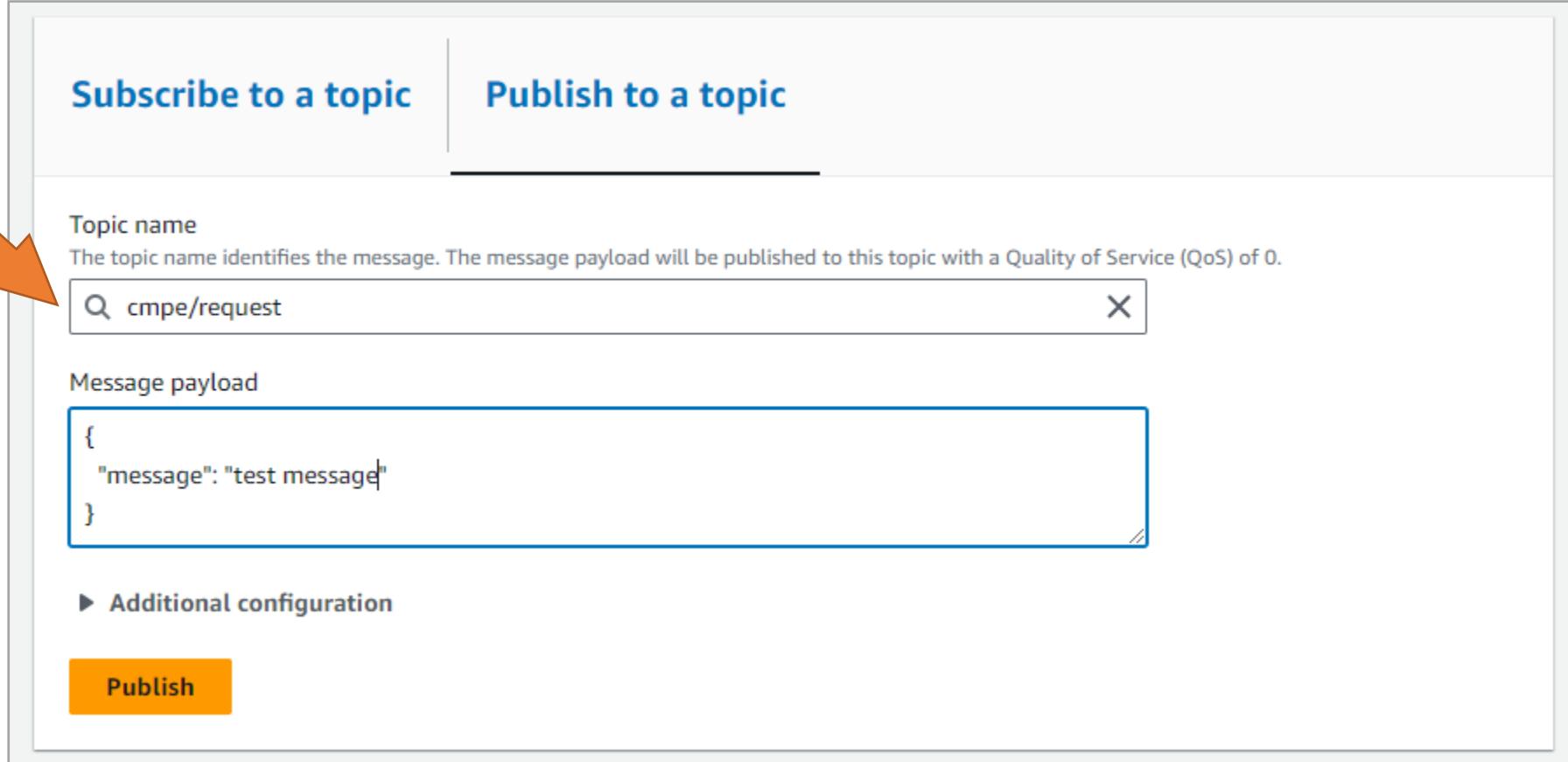
- Subscriptions table:** Shows one entry: "#". It includes a "Pause" button, a "Clear" button, an "Export" button, and an "Edit" button.
- Subscription details:** Shows the topic "#".
- Message publish dialog:** A modal window with an information icon and the text: "You cannot publish messages to a wildcard topic. Please select a different topic to publish messages to."

Publish message area:

- Text:** No messages have been sent to this subscription yet. Please send a message to this subscription to see messages here.

Test Your Deployment

Send a Test Message



The screenshot shows a user interface for publishing a message to an MQTT topic. At the top, there are two tabs: "Subscribe to a topic" and "Publish to a topic", with "Publish to a topic" being active. An orange arrow points to the "Topic name" input field, which contains the value "cmpe/request". Below the topic name is a "Message payload" section containing a JSON object: { "message": "test message" }. At the bottom left is a yellow "Publish" button.

Subscribe to a topic **Publish to a topic**

Topic name
The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

cmpe/request

Message payload

```
{  
  "message": "test message"  
}
```

► Additional configuration

Publish

Test Your Deployment

Check If the Client Responds Back

The screenshot shows the AWS IoT Core Subscriptions page. At the top, there is a yellow "Publish" button. Below it, a header row contains "Subscriptions" and "#". On the right side of this row are four buttons: "Pause", "Clear", "Export", and "Edit".

The main area displays two message entries:

- cmpe/response**: This entry was received on October 29, 2023, at 17:19:05 (UTC+0300). It contains the following JSON payload:

```
{  "timestamp": 1698589146478,  "response": "Hello from the Green Grass Core Device!"}
```
- cmpe/request**: This entry was received on October 29, 2023, at 17:19:05 (UTC+0300). It contains the following JSON payload:

```
{  "message": "Hello from AWS IoT console"}
```

Each message entry has a "Properties" button below it.

Other Use-Cases of AWS IoT Greengrass Core IPC

- Interact with component lifecycle.
 - Pause component, resume component
- Interact with component configuration.
 - Get and set component configuration parameters
- Retrieve secret values.
 - Gets the value of a secret that you store on the core device
- Authenticate and authorize client devices.
 - Verify the identity of a client device
 - Validates a client device's credentials
 - Verify whether a client device has permission to perform an action on a resource

AWS Lambda Functions on Core Devices

Run AWS Lambda Functions on Core Devices

- If you want to deploy an existing application code in Lambda functions to core devices, you can import AWS Lambda functions as components that run on AWS IoT Greengrass core devices.
- Lambda functions include dependencies on the following components.
 - The **Lambda launcher** component: handles processes and environment configuration.
 - The **Lambda manager** component: handles interprocess communication and scaling.
 - The **Lambda runtimes** component: provides artifacts for each supported Lambda runtime.
- You don't need to define these components as dependencies when you import the function.
- When you deploy the Lambda function component, the deployment includes these Lambda component dependencies.

Lambda Function Requirements

- A Linux-based OS with Java Runtime Environment (JRE) version 8 or greater.
- Minimum 256 MB disk space available for the AWS IoT Greengrass Core software.
- Minimum 96 MB RAM allocated to the AWS IoT Greengrass Core software.
- The `/tmp` directory must be mounted with exec permissions.
- Device must have the `mkfifo` shell command.
- Device must run the programming language libraries that a Lambda function requires.
 - Python, Node.js, and Java runtimes
- All of the following shell commands:
 - `ps -ax -o pid, ppid, sudo, sh, kill, cp, chmod, rm, ln, echo, exit, id, uname, grep`

Lambda Function Lifecycle

On-demand functions

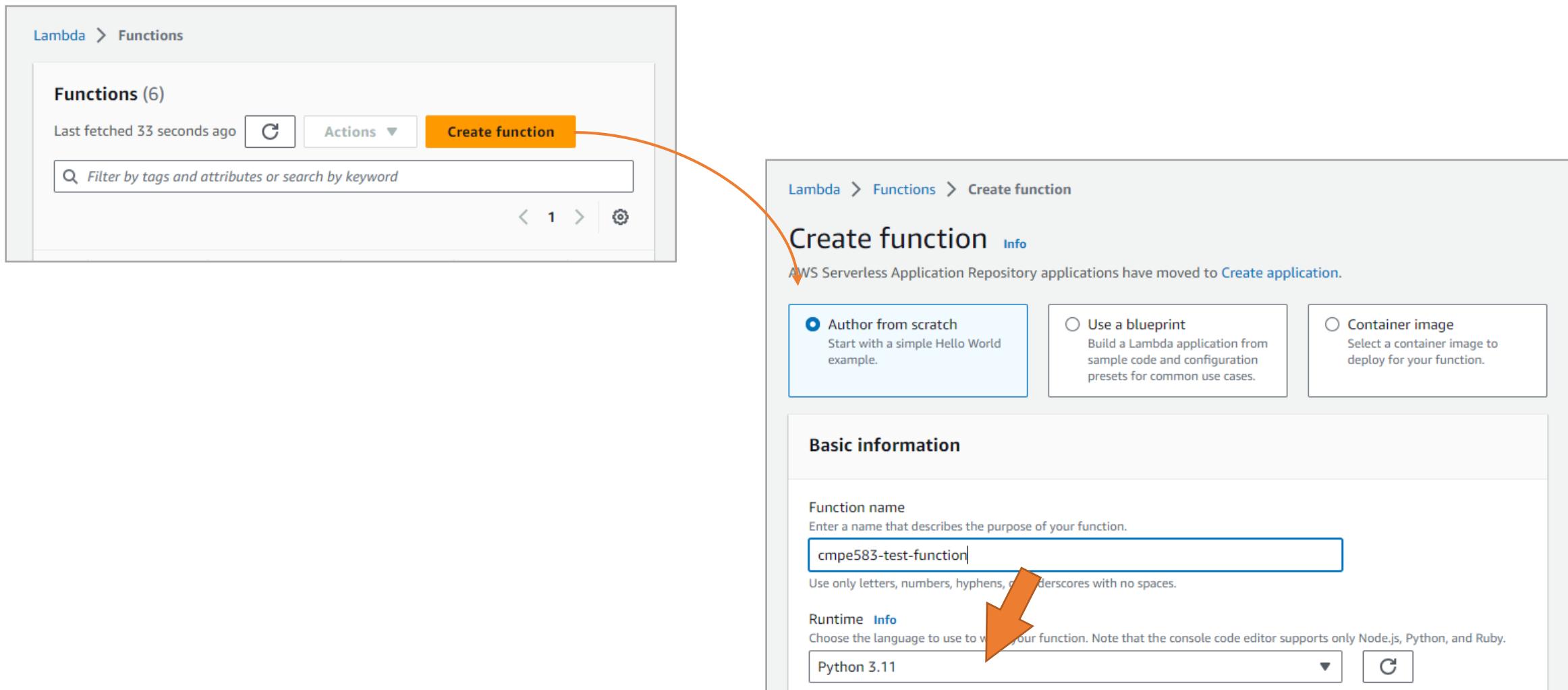
- On-demand functions start when they are invoked and stop when there are no tasks left to run.
- Each invocation of the function creates a separate container, also called a sandbox, to process invocations, unless an existing container is available for reuse.
- Any of the containers might process data that you send to the function.
- Multiple invocations of an on-demand function can run simultaneously.

Long-lived functions

- Long-lived (or pinned) functions start when the AWS IoT Greengrass Core software starts and run in a single container.
- The same container processes all data that you send to the function.
- Multiple invocations are queued until the AWS IoT Greengrass Core software runs earlier invocations.
- Use long-lived Lambda functions when you need to start doing work without any initial input.

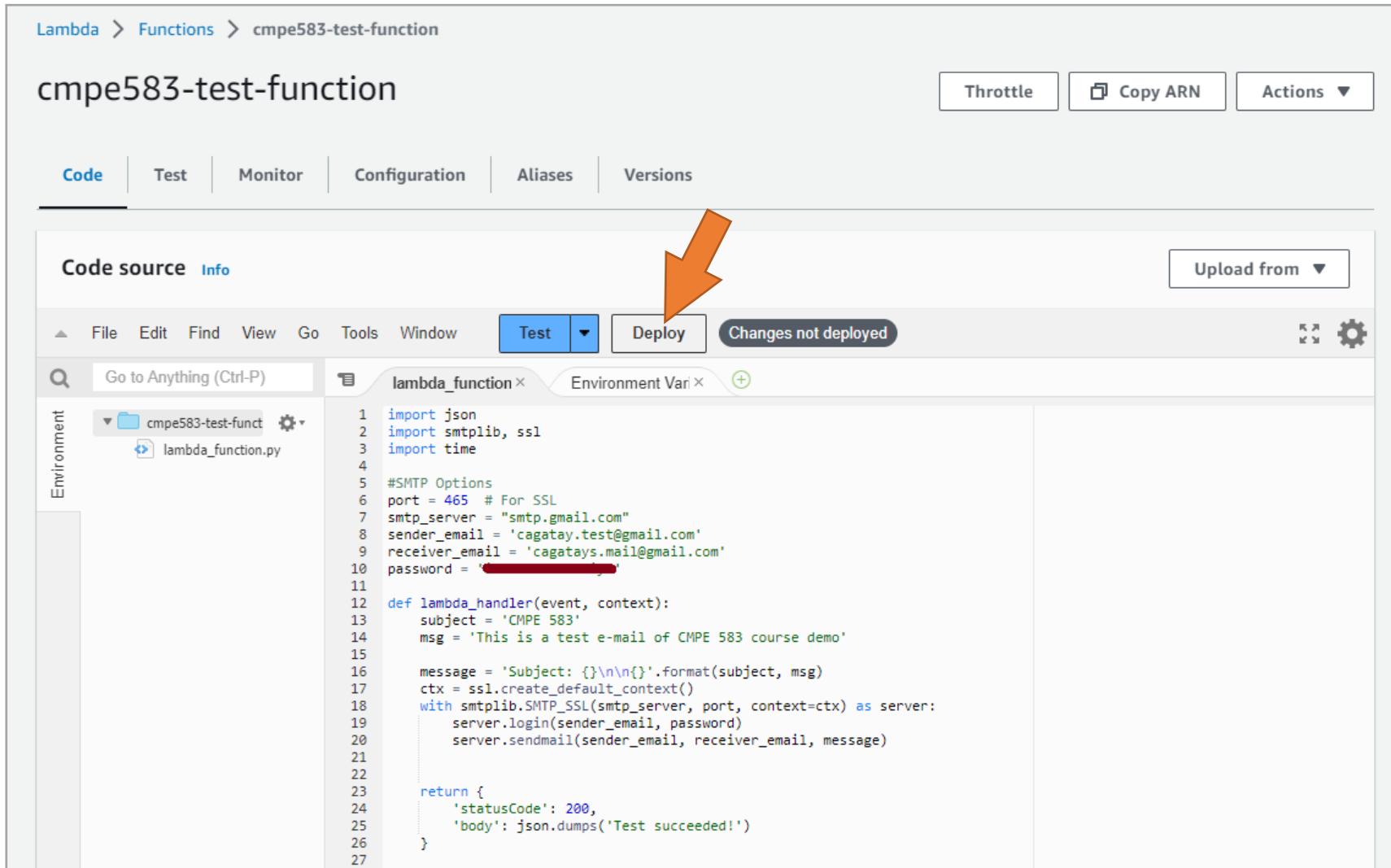
Step 1: Create AWS Lambda Function

Via AWS Lambda (Console)



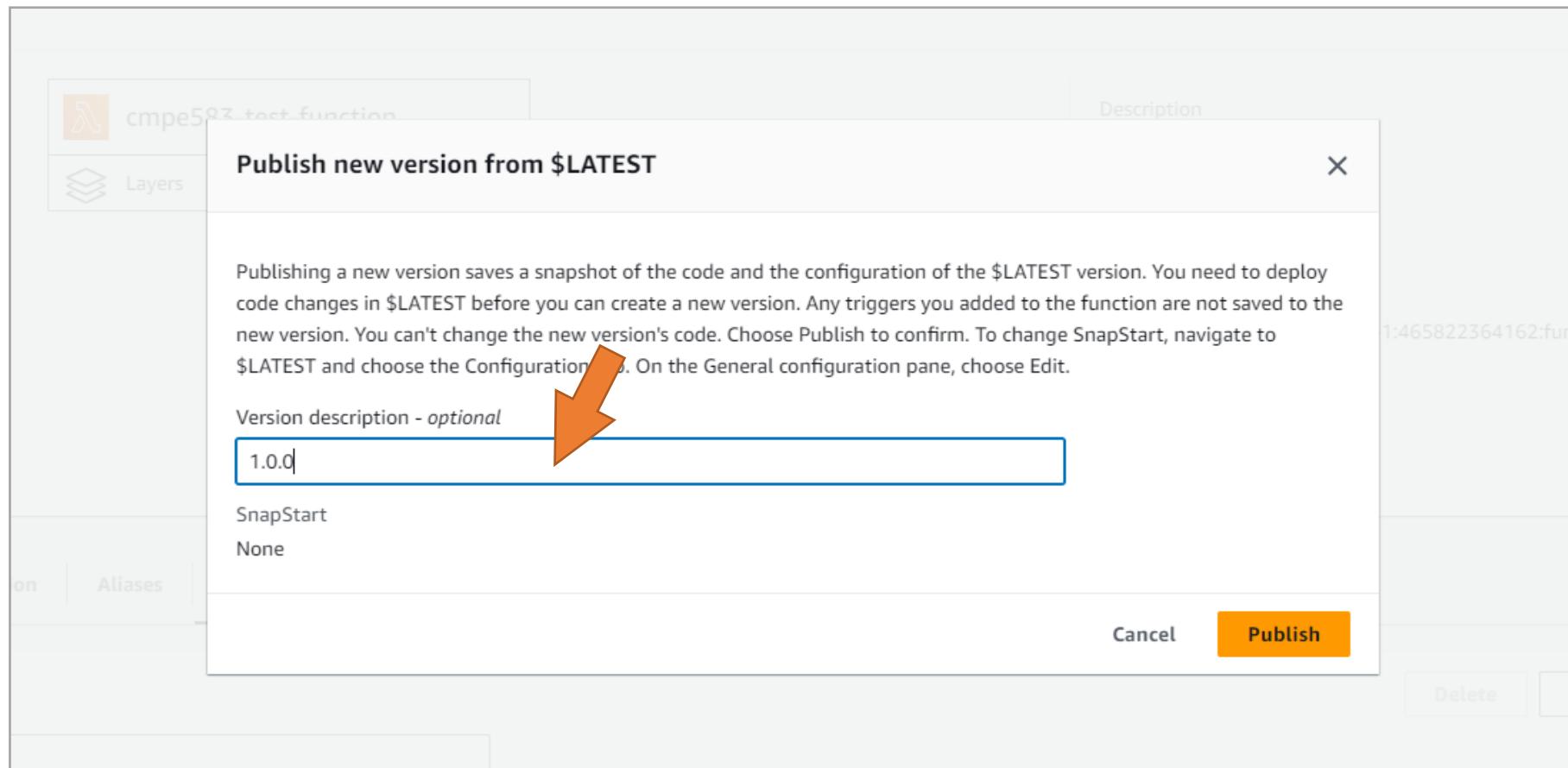
Step 1: Create AWS Lambda Function

Deploy the Lambda Function



Step 1: Create AWS Lambda Function

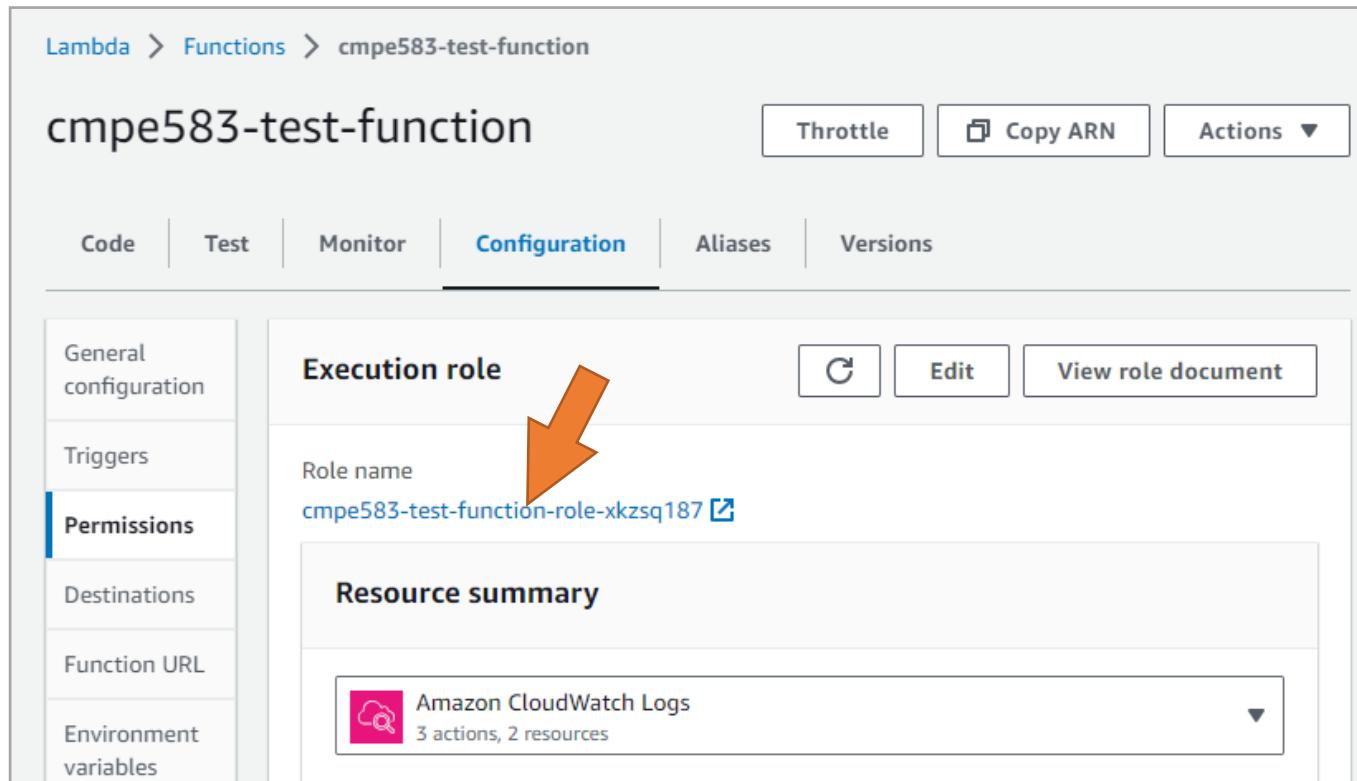
Publish the First Version



Step 1: Create AWS Lambda Function

Configure Lambda Role

- Our lambda function is very simple for testing purpose, but if you have a more complex function that needs to access AWS resources (server, S3 file, database, etc.), you will need to configure the lambda's role accordingly.



Step 2: Create a Custom Component

Import Lambda Function

The image consists of two screenshots illustrating the process of creating a custom component in AWS IoT Greengrass.

Screenshot 1: Greengrass components
This screenshot shows the 'Greengrass components' page. At the top, there are tabs for 'My components' (which is selected), 'Public components', and 'Community components'. Below this, a section titled 'My components (1)' displays a single private component. A prominent orange arrow points from the 'Create component' button (located below the search bar) to the 'Import Lambda function' section in the second screenshot.

Screenshot 2: Create component
This screenshot shows the 'Create component' wizard. It starts with a brief description: 'When you finish your component, you can add it to AWS IoT Greengrass to deploy to core devices. Provide the component recipe and artifacts to create the component. This component is private and visible only to your AWS account.'
The next section, 'Component information', contains instructions: '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.' A blue arrow points to the 'Import Lambda function' option.
The 'Component source' section offers three choices:

- 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.

The 'Lambda function' section allows selecting a Lambda function to import, with 'cmpe583-test-function' chosen and 'python3.11' as the runtime.
The 'Lambda function version' section lets you choose the function version.
The 'Component name - optional' section provides a field for a friendly name, with 'cmpe583.lambda.test' entered.
A note at the bottom specifies character restrictions: 'The name can be up to 128 characters. Valid characters: a-z, A-Z, 0-9, period (.), underscore (_), and hyphen (-)'.

Step 2: Create a Custom Component

Configure Event Source to Trigger Lambda Function

Lambda function configuration - optional

You can define default configuration options for the component that you create from the Lambda function. When you deploy this component, you can override these default values. [Learn more](#)

Event sources

Add event sources to subscribe your component for messages. The component can act on local publish/subscribe messages and AWS IoT Core MQTT messages.

| Topic | Type | Remove |
|-------------|-------------------|--------|
| cmpe/lambda | AWS IoT Core MQTT | |

[Add event source](#)

Timeout (seconds)

The maximum amount of time that the Lambda function can run before the AWS IoT Greengrass Core software stops it.

3

The timeout must be a positive integer.

Pinned

A pinned (long lived) Lambda function component starts when AWS IoT Greengrass starts and runs in its own container.

True
 False

► Additional parameters



Step 2: Create a Custom Component

Configure Lambda Lifecycle

The screenshot shows the 'Create component' dialog for a Lambda function. It includes two sections with orange arrows pointing to specific fields:

- Timeout (seconds)**: A text input field containing the value "3". An orange arrow points to the left side of this field.
- Pinned**: A section with a radio button labeled "False" which is selected. An orange arrow points to the left side of this section.
- Additional parameters**: A link to expand or collapse additional configuration options.
- Linux process configuration - optional**: A section describing default configuration options for the Linux process. It includes a note about overriding values during deployment.
- Isolation mode**: A dropdown menu currently set to "No container". An orange arrow points to the left side of this dropdown.

At the bottom right of the dialog are the "Cancel" and "Create component" buttons.

Step 3: Deploy Your Component

Add Component to Existing Deployment

The screenshot shows the AWS IoT Greengrass Components interface. On the left, there is a main panel titled "cmpe583.lambda.test" with a "Deploy" button highlighted in orange. A curved arrow points from this "Deploy" button to a modal window titled "Add to deployment".

The "Add to deployment" modal has two options: "Add to existing deployment" (selected) and "Create new deployment". It includes a search bar labeled "Find by deployment name or target name" and a table listing deployments. The table has columns: Deployment, Target name, Target type, and Status.

| Deployment | Target name | Target type | Status |
|---|-------------------------|-------------|--------|
| Test Deployment for CMPE583 | CMPE583-GreengrassGroup | Thing group | Active |

At the bottom of the modal are "Cancel" and "Next" buttons.

Step 3: Deploy Your Component

Configure Component to Merge Recipe

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 (4)

| Name | Version | Modified? |
|---|---------|-----------|
| cmpe583.lambda.test | 3.0.0 | - |
| samplePrintMsg | 1.0.0 | - |
| samplePubSub | 1.0.0 | - |
| aws.greengrass.clientdevices.IPDetector | 2.1.7 | - |

Configure component

Find by name

< 1 >

Cancel Skip to Review Previous Next

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. [Learn more](#)

```
1 ▼ {
2     "RecipeFormatVersion": "2020-01-25",
3     "ComponentName": "sampleLambda",
4     "ComponentVersion": "1.0.0",
5     "ComponentType": "aws.greengrass.generic",
6     "ComponentDescription": "A component that subscribes to a topic.",
7     "ComponentPublisher": "<Name>",
8     "ComponentConfiguration": {
9         "DefaultConfiguration": {
10            "accessControl": {
11                "aws.greengrass.ipc.mqttproxy": {
12                    "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
13                        "policyDescription": "Allows access to publish/subscribe to all topics.",
14                        "operations": [
15                            "aws.greengrass#PublishToIoTCore",
16                            "aws.greengrass#SubscribeToIoTCore"
17                        ],
18                        "resources": [
19                            "cmpe/lambda"
20                        ]
21                    }
22                }
23            }
24        }
25    }
26 }
```

Test Your Deployment

Via AWS MQTT Test Client

AWS IoT > MQTT test client

MQTT test client Info

You can use the MQTT test client to monitor the MQTT messages being published to inform devices and apps of changes and events. You can subscribe to topics and publish messages.

Subscribe to a topic **Publish to a topic**

Topic name
The topic name identifies the message. The message payload will be published to

Message payload
{
 "message": "test"
}

▶ Additional configuration

Publish

Gmail

Oluştur

Gelen Kutusu 41

- Yıldızlı
- Ertelenenler
- Önemli
- Gönderilmiş Postalar
- Taslaklar
- Kategoriler
- Sosyal

Postalarda arayın

CMPE 583 Gelen Kutusu

cagatay.test@gmail.com Alıcı: ▾

İngilizce ▾ → Türkçe ▾ İletiyi çevir

This is a test e-mail from CMPE 583 lecture!

Yanıtla Yönlendir

All Good!

Test Your Deployment

Verify From The Core Device's Logs

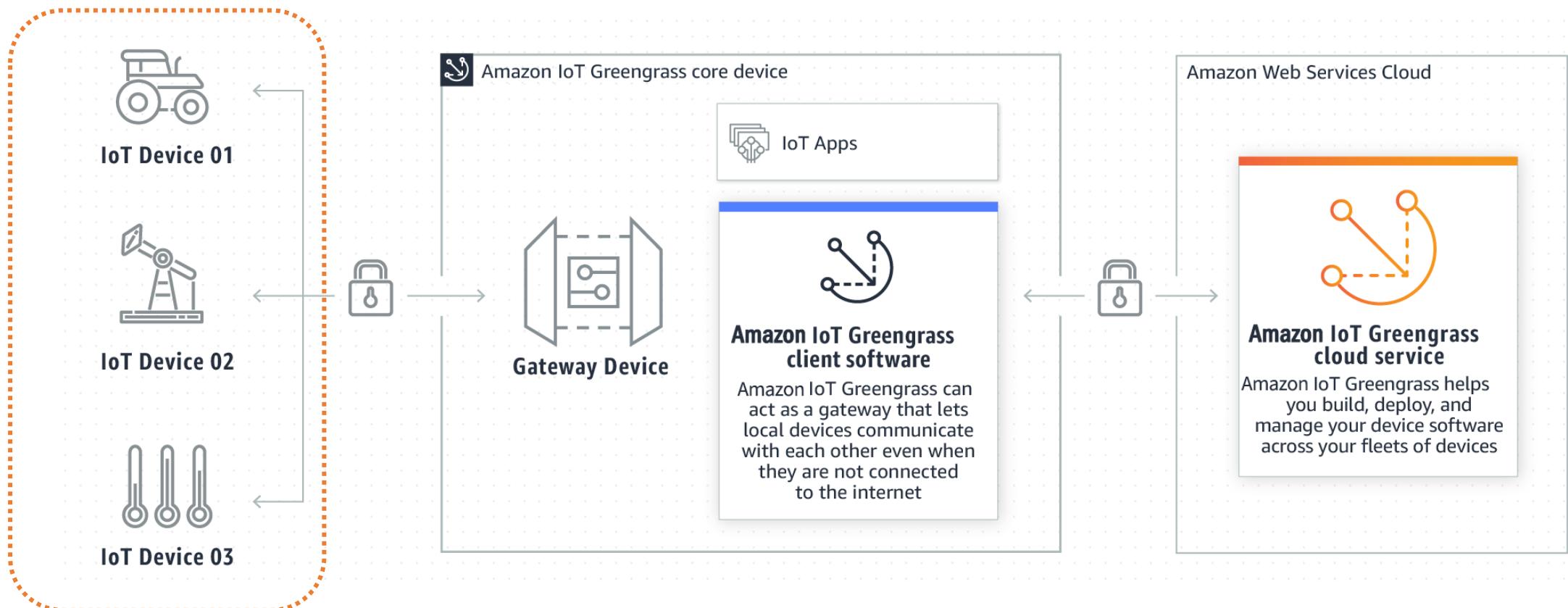
```
root@kubuntu:/home/cagatay/Projects/greengrass-demo# tail -n 100 -f /greengrass/v2/logs/cmpe583.lamda.test.log
2023-10-29T14:51:24.654Z [INFO] (pool-2-thread-52) cmpe583.lamda.test: shell-runner-start. {scriptName=services.cmpe583.lamda.test.lifecycle.startup.script, serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=STARTING, command=["/greengrass/v2/packages/artifacts/aws.greengrass.LambdaLauncher/2.0.12/lambda-..."]}
2023-10-29T14:51:24.821Z [INFO] (Copier) cmpe583.lamda.test: stdout. Started process: 3385. {scriptName=services.cmpe583.lamda.test.lifecycle.startup.script, serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=STARTING}
2023-10-29T14:51:24.824Z [INFO] (Copier) cmpe583.lamda.test: Startup script exited. {exitCode=0, serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=STARTING}
2023-10-29T14:51:24.924Z [INFO] (pool-2-thread-55) cmpe583.lamda.test: lambda_runtime.py:402,Status thread started. {serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=RUNNING}
2023-10-29T14:51:24.931Z [INFO] (pool-2-thread-55) cmpe583.lamda.test: lambda_runtime.py:154,Running [arn:aws:lambda:eu-central-1:465822364162:function:cmpe583-test-function:3]. {serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=RUNNING}
2023-10-29T14:51:25.052Z [INFO] (pool-2-thread-55) cmpe583.lamda.test: lambda_function.py:13,Sending test email... {serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=RUNNING}
2023-10-29T14:51:26.757Z [INFO] (pool-2-thread-55) cmpe583.lamda.test: lambda_function.py:24,test email was sent!. {serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=RUNNING}
2023-10-29T14:51:54.581Z [INFO] (pool-2-thread-60) cmpe583.lamda.test: shell-runner-start. {scriptName=services.cmpe583.lamda.test.lifecycle.shutdown.script, serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=STOPPING, command=["/greengrass/v2/packages/artifacts/aws.greengrass.LambdaLauncher/2.0.12/lambda-..."]}
2023-10-29T14:51:54.740Z [INFO] (pool-2-thread-55) cmpe583.lamda.test: lambda_runtime.py:370,Caught signal 15. Stopping runtime.. {serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=STOPPING}
```

Local IoT (Client) Devices



Local IoT Devices

- **Client devices** are local IoT devices that connect to and communicate with a Greengrass **core device** over MQTT.



Client Device and Core Device Communication

Use -Cases

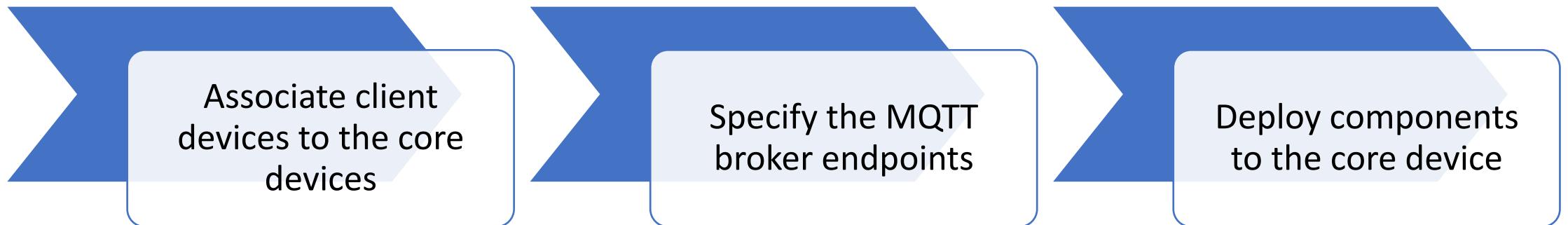
- You can connect client devices to core devices to do various operations:
 - Interact with MQTT messages in Greengrass components.
 - Relay messages and data between client devices and AWS IoT Core.
 - Interact with client device shadows in Greengrass components.
 - Sync client devices shadows with AWS IoT Core.
- AWS IoT Greengrass provides some public components that you can deploy to core devices.
- These components enable client devices to connect and communicate with a core device.

AWS-Provided Client Device Components

| Component | Description | Depends on nucleus | Component type | Supported OS | Open source |
|------------------------------|---|--------------------|----------------|----------------|---------------------|
| Client device auth | Enables local IoT devices, called client devices, to connect to the core device. | Yes | Plugin | Linux, Windows | Yes |
| IP detector | Reports MQTT broker connectivity information to AWS IoT Greengrass, so client devices can discover how to connect. | Yes | Plugin | Linux, Windows | Yes |
| MQTT bridge | Relays MQTT messages between client devices, local AWS IoT Greengrass publish/subscribe, and AWS IoT Core. | No | Plugin | Linux, Windows | Yes |
| MQTT 3.1.1 broker (Moquette) | Runs an MQTT 3.1.1 broker that handles messages between client devices and the core device. | No | Plugin | Linux, Windows | Yes |
| MQTT 5 broker (EMQX) | Runs an MQTT 5 broker that handles messages between client devices and the core device. | No | Generic | Linux, Windows | No |
| Shadow manager | Enables interaction with shadows on the core device. It manages shadow document storage and also the synchronization of local shadow states with the AWS IoT Device Shadow service. | Yes | Plugin | Linux, Windows | Yes |

Cloud Discovery Configuration

- You should configure cloud discovery to connect client devices to core devices.
- When you configure cloud discovery, client devices can connect to the AWS IoT Greengrass cloud service to retrieve information about core devices.
- Then, the client devices can attempt to connect to each core device until they successfully connect.
- To use cloud discovery, you must do the following:



Create Client Devices

Step 1: Configure the Greengrass Service Role

Verify If Greengrass Service Role is Attached

- The Greengrass service role is an IAM service role that authorizes AWS IoT Greengrass to access resources from AWS services on your behalf.
- Check whether the Greengrass service role is set up in AWS IoT console's settings.

The screenshot shows the AWS IoT Greengrass service role configuration page. On the left, there is a sidebar with links: Device software, Billing groups, Settings (which is selected), Feature spotlight, and Documentation. At the bottom of the sidebar is a link to 'Tell us what you think'. The main content area has a title 'Greengrass service role' with an 'Info' link, a 'Detach role' button, and a 'Change role' button. Below the title, it says 'AWS IoT Greengrass works with other AWS services, such as AWS IoT and AWS Lambda.' 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.' Another note says: 'If you have a service role that's already defined, you can attach it. Otherwise, you must create one first.' Below this is a section for 'Current service role' showing 'arn:aws:iam::465822364162:role/CMPE583-GreengrassRole'. To the right, there is a section for 'Policies attached to this role' showing 'AWSGreengrassResourceAccessRolePolicy'. At the bottom right of the page is a footer: 'Sensitivity: Internal / Non-Personal Data'.

Step 2: Configure the AWS IoT Thing Policy

Via AWS IoT Console

The screenshot shows the AWS IoT Things page with the following steps highlighted:

1. AWS IoT sidebar: Shows the navigation menu with "Things" selected.
2. AWS IoT sidebar: Shows the navigation menu with "Things" selected.
3. Thing details: The thing name is CMPE583-GreengrassCore.
4. Certificates tab: The "Certificates" tab is selected.
5. Certificate ID: The certificate ID is c62e9f5e5a6932223537f9fb3c477368dd8f379f45a930bf7b0285bb91ffc3ac.

Thing details

| | |
|---|--------------------|
| Name CMPE583-GreengrassCore | Type - |
| ARN arn:aws:iot:eu-central-1:4658223...62:thing/CMPE583-GreengrassCore | Billing group - |

Attributes Certificates Thing groups Device Shadows Activity Packages and versions Jobs Alarms De...

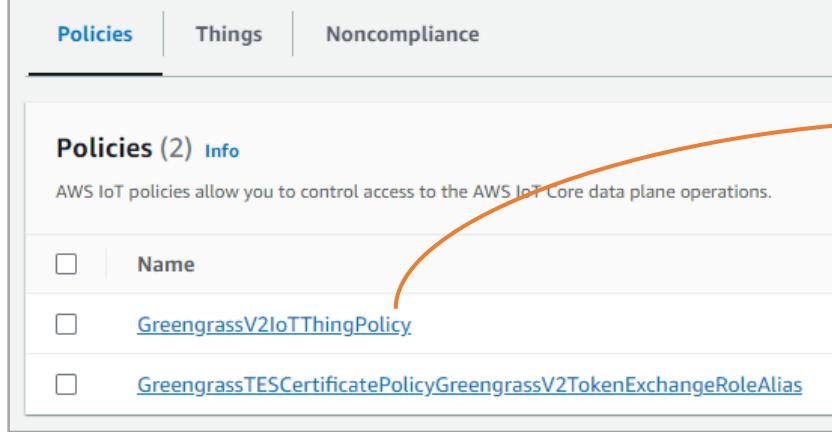
Certificates (1) Info

The device certificates attached to this thing resource.

| Find certificates | Detach | Create certificate |
|--|--|--------------------|
| <input type="checkbox"/> Certificate ID | <input checked="" type="checkbox"/> Status | |
| c62e9f5e5a6932223537f9fb3c477368dd8f379f45a930bf7b0285bb91ffc3ac | Active | |

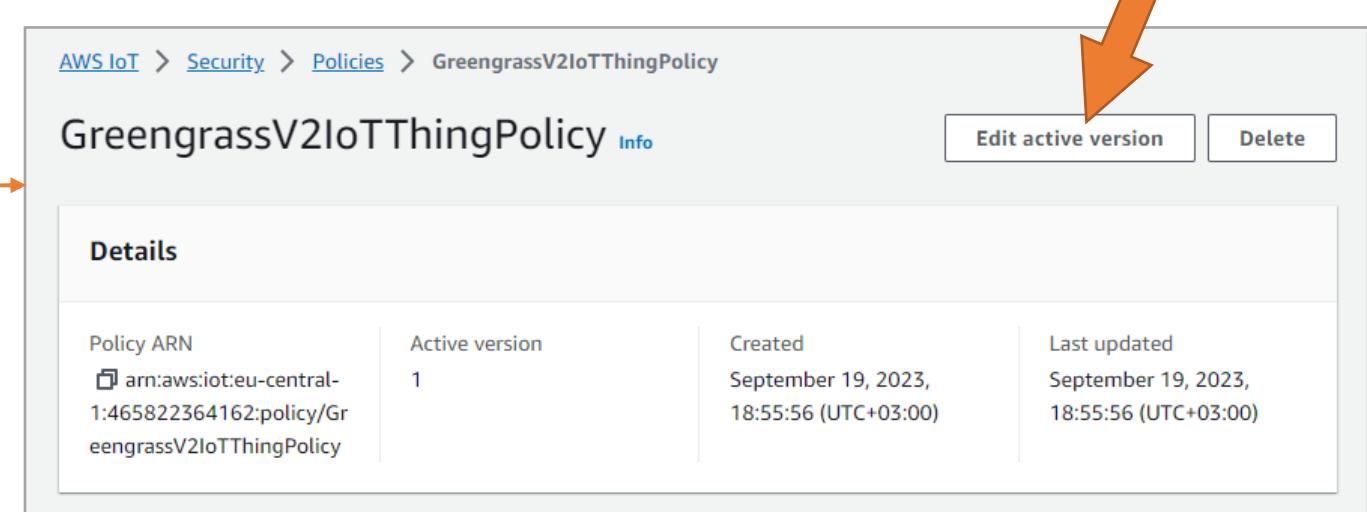
Step 2: Configure the AWS IoT Thing Policy

Review the Policy



AWS IoT Policies list:

- Name: GreengrassV2IoTThingPolicy
- Name: GreengrassTESCertificatePolicyGreengrassV2TokenExchangeRoleAlias



AWS IoT Policies > Security > Policies > GreengrassV2IoTThingPolicy

GreengrassV2IoTThingPolicy Info

Details

| Policy ARN | Active version | Created | Last updated |
|---|----------------|--|--|
| arn:aws:iot:eu-central-1:465822364162:policy/GreengrassV2IoTThingPolicy | 1 | September 19, 2023, 18:55:56 (UTC+03:00) | September 19, 2023, 18:55:56 (UTC+03:00) |

Edit active version **Delete**

- Review the policy for required permissions, and add any required permissions that are missing:
 - greengrass:PutCertificateAuthorities
 - greengrass:VerifyClientDeviceIdentity
 - greengrass:VerifyClientDeviceIoTCertificateAssociation
 - greengrass:GetConnectivityInfo

Step 2: Configure the AWS IoT Thing Policy

Allow Core Device to Sync Shadows

AWS IoT > Security > Policies > GreengrassV2IoTThingPolicy > Edit

Edit policy: GreengrassV2IoTThingPolicy (Version 1)

Policy statements | Policy examples

Policy document Info

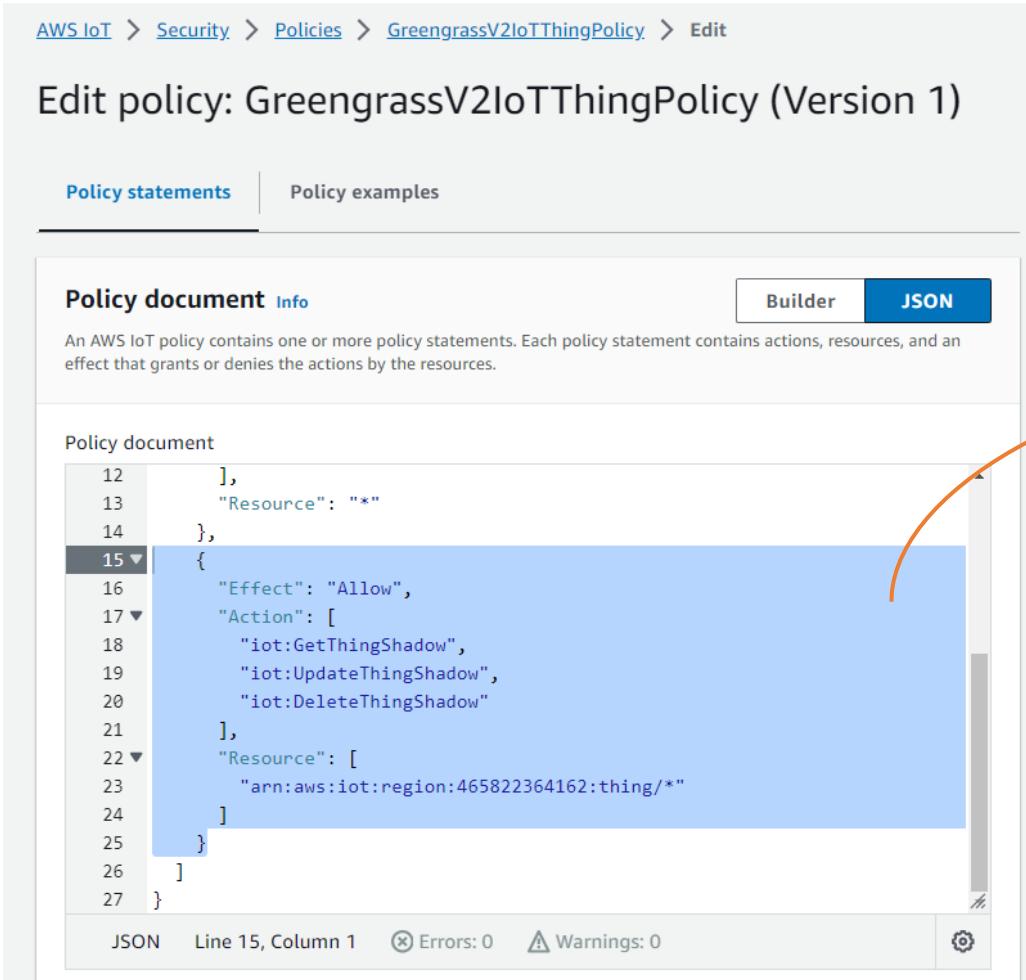
An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy document

```
12     ],
13     "Resource": "*"
14   },
15   {
16     "Effect": "Allow",
17     "Action": [
18       "iot:GetThingShadow",
19       "iot:UpdateThingShadow",
20       "iot:DeleteThingShadow"
21     ],
22     "Resource": [
23       "arn:aws:iot:region:465822364162:thing/*"
24     ]
25   }
26 ]
27 }
```

Builder JSON

JSON Line 15, Column 1 Errors: 0 Warnings: 0



To allow the core device to sync shadows with AWS IoT Core, add the following statement to the policy:

```
{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot:DeleteThingShadow" ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/*"
  ]
}
```

Step 2: Configure the AWS IoT Thing Policy

Update & Verify Policy

The screenshot illustrates the process of updating and verifying an AWS IoT Thing Policy. It consists of two main panels: a left panel showing policy versions and a right panel showing the active policy's details.

Left Panel: All versions (1/2) Info

- Buttons:** C (Create), Delete, Set as active (highlighted with an orange arrow), Edit version, View JSON.
- Text:** The active and previous versions of this policy. Only one version can be active. A policy can have no more than 5 versions. To update a policy with 5 versions, you must first delete one.
- Table:** Shows two versions:
 - Version 2:** Inactive, Created November 04, 2023, 14:... (selected).
 - Version 1:** Active, Created September 19, 2023, 18:... (unchecked).

Right Panel: Active version: 2 Info

- Buttons:** Builder (highlighted with an orange arrow), JSON.
- Table:** Displays the policy's effect, actions, and resources.

| Policy effect | Policy action | Policy resource |
|---------------|-----------------------|---|
| Allow | iot:Connect | * |
| Allow | iot:Publish | * |
| Allow | iot:Subscribe | * |
| Allow | iot:Receive | * |
| Allow | greengrass:* | * |
| Allow | iot:GetThingShadow | arn:aws:iot:region:465822364162:thing/* |
| Allow | iot:UpdateThingShadow | arn:aws:iot:region:465822364162:thing/* |
| Allow | iot:DeleteThingShadow | arn:aws:iot:region:465822364162:thing/* |

Step 3: Create Client Device (Thing)

Via AWS IoT Console

The screenshot shows the AWS IoT Things management interface. On the left, a sidebar menu lists 'Manage' options: 'All devices' (selected), 'Things' (highlighted with an orange arrow), 'Thing groups', 'Thing types', 'Fleet metrics', 'Greengrass devices' (selected), 'Core devices', 'Components', 'Deployments', and 'Groups (V1)'. The main content area is titled 'AWS IoT > Manage > Things' and shows 'Things (2) Info'. It includes a search bar, a table of two items (CMPE583-GreengrassCore and MyFirstIOTThing), and buttons for 'Advanced search', 'Run aggregations', 'Edit', 'Delete', and a prominent orange 'Create things' button. An orange arrow points to the 'Create things' button.

| Name | Thing type |
|------------------------|------------|
| CMPE583-GreengrassCore | - |
| MyFirstIOTThing | - |

Step 3: Create Client Device (Thing)

Via AWS IoT Console

AWS IoT > Manage > Things > Create things

Create things Info

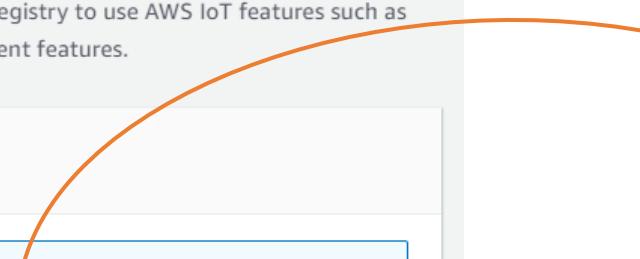
A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Number of things to create

Create single thing
Create a thing resource to register a device. Provision the certificate and policy necessary to allow the device to connect to AWS IoT.

Create many things
Create a task that creates multiple thing resources to register devices and provision the resources those devices require to connect to AWS IoT.

Cancel **Next**



AWS IoT > Manage > Things > Create things > Create single thing

Step 1 of 3

Specify thing properties Info

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Thing properties Info

Thing name

CMPE_Test_Client1

Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

Step 3: Create Client Device (Thing)

Configure Certificate & Policy

AWS IoT > Manage > Things > Create things > Create single thing

Step 2 of 3

Configure device certificate - optional Info

A device requires a certificate to connect to AWS IoT. You can choose how to register a certificate for your device now, or you can create and register a certificate for your device later. Your device won't be able to connect to AWS IoT until it has an active certificate with an appropriate policy.

Device certificate

Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

AWS IoT > Manage > Things > Create things > Create single thing

Step 3 of 3

Attach policies to certificate - optional Info

AWS IoT policies grant or deny access to AWS IoT resources. Attaching policies to the device certificate applies this access to the device.

Policies (1/4)

Select up to 10 policies to attach to this certificate.

Filter policies

< 1 > Settings

| Name |
|---|
| <input type="checkbox"/> MyIOTPolicyVeryExtensive2 |
| <input type="checkbox"/> MyFirstIOTThing-Policy |
| <input checked="" type="checkbox"/> GreengrassV2IoTThingPolicy |
| <input type="checkbox"/> GreengrassTESCertificatePolicyGreengrassV2TokenExchangeRoleAlias |

Cancel Previous **Create thing**

Step 3: Create Client Device (Thing)

Download Key Files

Download certificates and keys

Download certificate and key files to install on your device so that it can connect to AWS.

Device certificate

You can activate the certificate now, or later. The certificate must be active for a device to connect to AWS IoT.

Device certificate
42f288d1e64...te.pem.crt

[Deactivate certificate](#) [Download](#)

Key files

The key files are unique to this certificate and can't be downloaded after you leave this page. Download them now and save them in a secure place.

⚠ This is the only time you can download the key files for this certificate.

Public key file
42f288d1e642ddd6b7e2f0c...ae9c731-public.pem.key

[Download](#) **Key downloaded**

Private key file
42f288d1e642ddd6b7e2f0c...e9c731-private.pem.key

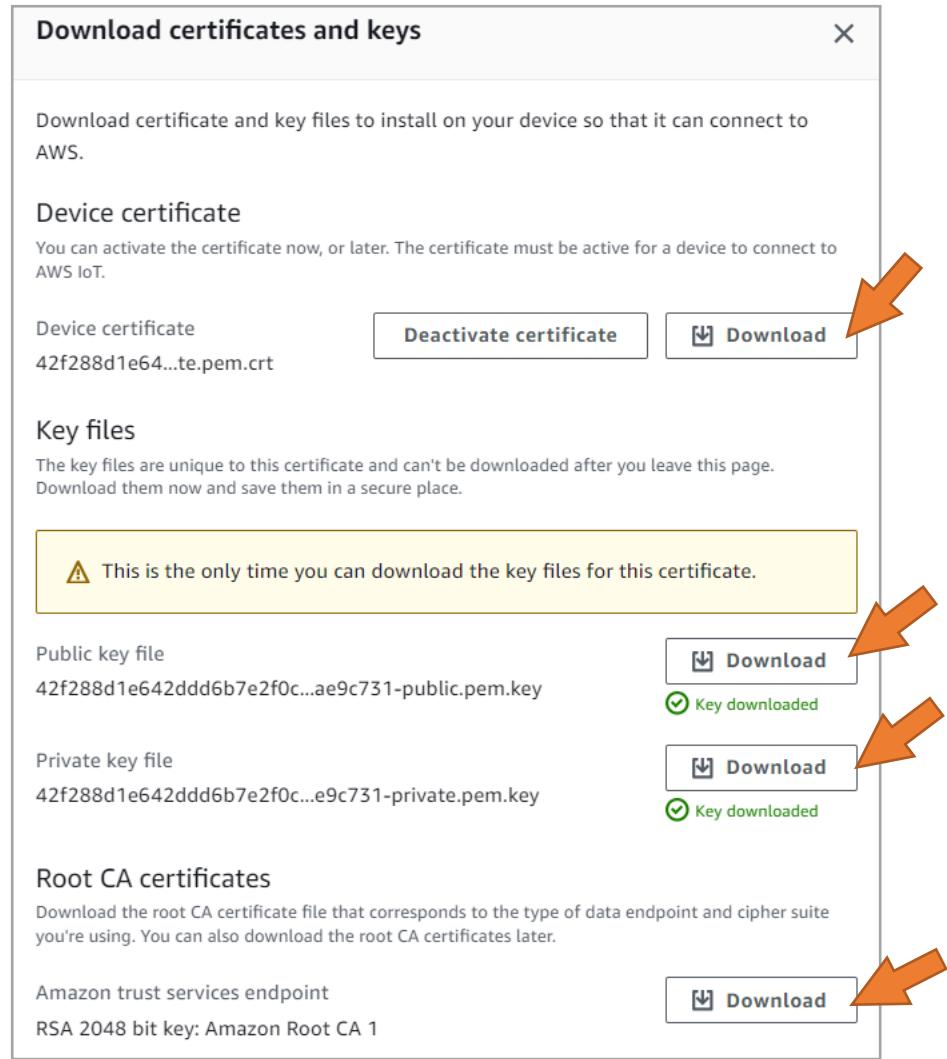
[Download](#) **Key downloaded**

Root CA certificates

Download the root CA certificate file that corresponds to the type of data endpoint and cipher suite you're using. You can also download the root CA certificates later.

Amazon trust services endpoint
RSA 2048 bit key: Amazon Root CA 1

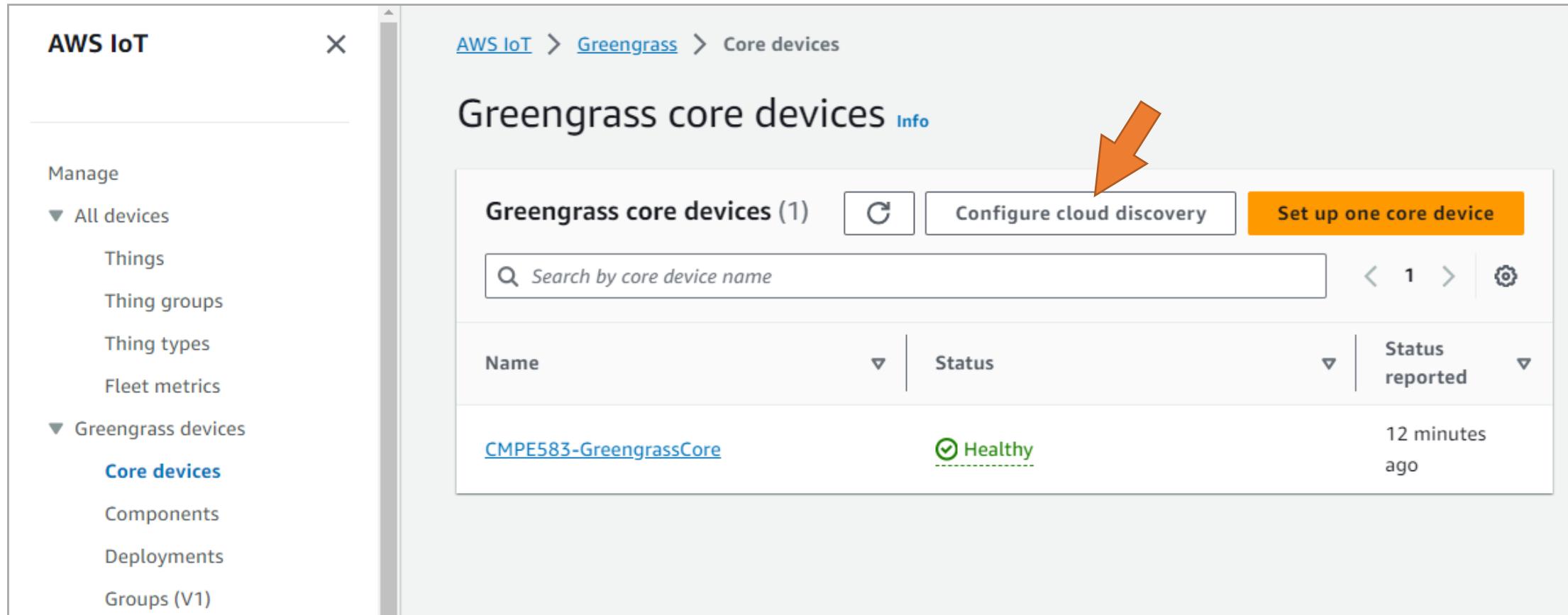
[Download](#)



★ Key files must be downloaded in the last step of things creation!

Step 4: Enable Client Device Support

Configure Cloud Discovery



The screenshot shows the AWS IoT Greengrass Core devices interface. On the left, there's a sidebar with 'Manage' and 'Greengrass devices' sections. Under 'Greengrass devices', 'Core devices' is selected. The main area displays 'Greengrass core devices (1)'. A table lists one device: 'CMPE583-GreengrassCore' with a status of 'Healthy' (indicated by a green checkmark). Above the table, there are three buttons: a refresh icon, 'Configure cloud discovery' (which has an orange arrow pointing to it), and 'Set up one core device'. The URL in the browser header is 'AWS IoT > Greengrass > Core devices'.

| Name | Status | Reported |
|------------------------|---------|----------------|
| CMPE583-GreengrassCore | Healthy | 12 minutes ago |

Step 4: Enable Client Device Support

Select Target Core Devices

Configure core device discovery Info

Configure core devices to securely communicate with local IoT devices, called client devices, over MQTT. On this page, you configure how client devices discover your core device. You also choose which Greengrass components to deploy to your core device to enable client devices to connect, interact with the core device, and sync with the AWS Cloud.

Step 1: Select target core devices

Specify a core device, or a thing group that contains core devices, to configure discovery. This process creates a deployment to the core device or thing group that you specify.

Target type
 Core device
 Thing group

Target name
CMPE583-GreengrassCore

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

You can configure core device discovery for a code device or a thing group.

Step 4: Enable Client Device Support

Associate Client Devices

Step 2: Associate client devices Info

With cloud discovery, you associate AWS IoT things as client devices. Use the Greengrass discovery client in the AWS IoT Device SDK to connect client devices to core devices.

Create AWS IoT things to represent client devices

You can create AWS IoT things to associate as client devices. Use the AWS IoT console to create a thing and download security certificates to your client device.

[Create AWS IoT thing](#)

Associated client devices (1) Info

Manage which client devices (AWS IoT things) associate with this core device. Associated client devices can discover this core device's endpoints using cloud discovery.

[Find by thing name](#)

[Disassociate](#) [Associate client devices](#) [C](#)

| Name | Client device associated |
|-------------------|--------------------------|
| CMPE_Test_Client1 | |

1

Associate client devices with core device

Specify one or more AWS IoT things to associate as Greengrass client devices.

AWS IoT thing name

CMPE_Test_Client1 [Add](#) [View AWS IoT things](#)

2

Cancel [Associate](#)

3

Step 4: Enable Client Device Support

Deploy Required Greengrass Components

- The following Greengrass components should be deployed to the core device to enable client devices to connect and communicate with a core device:
 - Client device auth (`aws.greengrass.clientdevices.Auth`)
 - MQTT 3.1.1 broker (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)
 - MQTT 5 broker (EMQX) (`aws.greengrass.clientdevices.mqtt.EMQX`)
- The following components are optional:
 - MQTT bridge (`aws.greengrass.clientdevices.mqtt.Bridge`)
 - IP detector (`aws.greengrass.clientdevices.IPDetector`)
 - Shadow manager (`aws.greengrass.ShadowManager`)

Step 4: Enable Client Device Support

Configure Required Greengrass Components

Step 3: Configure and deploy Greengrass components

Greengrass core devices require a set of components to support client devices. Select and configure the components to deploy. Then, choose **Review and deploy** to open the deployment page, where you create a new deployment or revise the latest deployment for the core device target.

Greengrass nucleus - aws.greengrass.Nucleus
This component is the minimum installation of the AWS IoT Greengrass Core software, which every core device runs. Client device support requires nucleus v2.2.0 or later.
[Edit configuration](#)

Client device auth - aws.greengrass.clientdevices.Auth Info
Deploy this component to authenticate client devices and authorize client device actions. Configure this component to specify which client devices can connect and what actions they can perform.
[Edit configuration](#)

MQTT 3.1.1 broker (Moquette) - aws.greengrass.clientdevices.mqtt.Moquette Info
Deploy this component to use the Moquette MQTT broker, which is compliant with the MQTT 3.1.1 standard. Choose this option for a lightweight MQTT broker. You can configure the port that the MQTT broker uses. The default port is port 8883.
[Edit configuration](#)

MQTT 5 broker (EMQX) - aws.greengrass.clientdevices.mqtt.EMQX Info
Deploy this component to use the EMQX MQTT broker, which is compliant with the MQTT 5 standard. Choose this option to use MQTT 5 features in communication between client devices and the core device. You can configure the port that the MQTT broker uses. The default port is port 8883. This component requires Greengrass nucleus v2.6.0 or later.
[Edit configuration](#)

MQTT bridge - aws.greengrass.clientdevices.mqtt.Bridge Info
Deploy this component to relay MQTT messages between client devices, local publish/subscribe, and the AWS IoT Core cloud service. Configure this component to specify which topics to relay.
[Edit configuration](#)

IP detector - aws.greengrass.clientdevices.IPDetector Info
Deploy this component to manage the core device's MQTT endpoints in the AWS IoT Greengrass cloud service, so client devices know where to connect. You can use this component if you have a simple network setup or client devices on the same network as the core device. Otherwise, you can manually manage the core device's endpoints.
[Edit configuration](#)

Shadow manager - aws.greengrass.ShadowManager Info
Deploy this component to enable the local shadow service, which enables you to interact with and sync client device shadows. You must configure the MQTT bridge component to relay messages between client devices and shadow manager, which uses local publish/subscribe. Client device shadow support requires Greengrass nucleus v2.6.0 or later, shadow manager v2.2.0 or later, and MQTT bridge v2.2.0 or later.
[Edit configuration](#)

MQTT bridge and client device auth components should be configured before the deployment!

★ It is recommended to deploy only one MQTT broker component. If you deploy multiple MQTT broker components, you must configure them to use different ports!

Step 4: Enable Client Device Support

Customize aws.greengrass.clientdevices.Auth

Client device auth - aws.greengrass.clientdevices.Auth [Info](#)

Deploy this component to authenticate client devices and authorize client device actions. Configure this component to specify which client devices can connect and what actions they can perform.

[Edit configuration](#)

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. [Learn more](#)

```
1 ▼ {
2   "deviceGroups": {
3     "formatVersion": "2021-03-05",
4     "definitions": {
5       "MyPermissiveDeviceGroup": {
6         "selectionRule": "thingName: *",
7         "policyName": "MyPermissivePolicy"
8       }
9     },
10    "policies": {
11      "MyPermissivePolicy": {
12        "AllowAll": {
13          "statementDescription": "Allow clients to perform all actions",
14          "operations": [
15            "*"
16          ],
17          "resources": [
18            "*"
19          ]
20        }
21      }
22    }
23  }
```

JSON Ln 25, Col 1 Errors: 0 Warnings: 0

In this example, permissive policy is used to allow all operations. To use a more restrictive policy, check out the policy file in github:
cmpe583-ClientDeviceAuth.Json

Step 4: Enable Client Device Support

Customize aws.greengrass.clientdevices.mqtt.Bridge

MQTT bridge - aws.greengrass.clientdevices.mqtt.Bridge [Info](#)

Deploy this component to relay MQTT messages between client devices, local publish/subscribe, and the AWS IoT Core cloud service. Configure this component to specify which topics to relay.

[Edit configuration](#)

- Relay MQTT messages on the clients/+/hello/world topic filter from client devices to the AWS IoT Core cloud service.
- For example, this topic filter matches the clients/CMPE_Test_Client1/hello/world topic.

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. [Learn more](#)

```
1 ▼ {  
2 ▼   "mqttTopicMapping": {  
3 ▼     "ClientDeviceHelloWorld": {  
4       "topic": "clients/+/hello/world",  
5       "source": "LocalMqtt",  
6       "target": "IotCore"  
7     }  
8   }  
9 }  
10 |
```

JSON Ln 10, Col 1 ✘ Errors: 0 ⚠ Warnings: 0

Step 4: Enable Client Device Support

Review & Deploy Greengrass Components

MQTT bridge - aws.greengrass.clientdevices.mqtt.Bridge [Info](#)
Deploy this component to relay MQTT messages between client devices, local publish/subscribe, and the AWS IoT Core cloud service. Configure this component to specify which topics to relay.

[Edit configuration](#)

IP detector - aws.greengrass.clientdevices.IPDetector [Info](#)
Deploy this component to manage the core device's MQTT endpoints in the AWS IoT Greengrass cloud service, so client devices know where to connect. You can use this component if you have a simple network setup or client devices on the same network as the core device. Otherwise, you can manually manage the core device's endpoints.

[Edit configuration](#)

Shadow manager - aws.greengrass.ShadowManager [Info](#)
Deploy this component to enable the local shadow service, which enables you to interact with and sync client device shadows. You must configure the MQTT bridge component to relay messages between client devices and shadow manager, which uses local publish/subscribe. Client device shadow support requires Greengrass nucleus v2.6.0 or later, shadow manager v2.2.0 or later, and MQTT bridge v2.2.0 or later.

[Edit configuration](#)

[Cancel](#) [Review and deploy](#)



Step 4: Enable Client Device Support

Check Deployment Status

Deployment successfully created

Your configuration is now being deployed. After the deployment successfully completes, you can connect client devices to the core device. To test communication, follow instructions in our documentation. [Learn more](#)

AWS IoT > Greengrass > Deployments > Deployment for CMPE583-GreengrassCore

Deployment for CMPE583-GreengrassCore

Latest revision: 1 ▾ Cancel Actions ▾

| Overview | | |
|----------------------------------|--------------------------------|------------------------------------|
| Target CMPE583-GreengrassCore | Target type Core device | Deployment created 1 minute ago |
| Device status Healthy | Deployment status Completed | |



Connect Client Devices

Step 1: Install AWS IoT Device SDKs

Programming Languages

- Client devices can use the AWS IoT Device SDK to discover, connect, and communicate with a core device.
- AWS IoT Device SDKs
 - AWS IoT C++ Device SDK
 - AWS IoT Device SDK for Python
 - AWS IoT Device SDK for JavaScript
 - AWS IoT Device SDK for Java
- AWS Mobile SDKs
 - AWS Mobile SDK for Android
 - AWS Mobile SDK for iOS

Step 1: Install AWS IoT Device SDKs

AWS IoT Device SDK v2 for Python

- AWS IoT Device SDK is open source and publicly available in GitHub!

Clone the AWS IoT Device SDK v2 for Python repository to download it.

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

Install the AWS IoT Device SDK v2 for Python.

```
python3 -m venv venvclient  
source venvclient/bin/activate
```

```
(venvclient)$ python3 -m pip install ./aws-iot-device-sdk-python-v2
```

★ It is recommended to install & run aws-iot-device-sdk in virtual environment to avoid making any changes that affect the entire system!

Step 2: Run Sample Device Discovery

With samples/basic_discovery.py

- ✓ The discovery sample application sends the message 10 times and disconnects. It also subscribes to the same topic where it publishes messages.

```
(venvclient)$ cd aws-iot-device-sdk-python-v2/samples  
  
(venvclient)$ python3 basic_discovery.py \  
--thing_name CMPE_Test_Client1 \  
--topic 'clients/CMPE_Test_Client1/hello/world' \  
--message 'Hello World!' \  
--ca_file ~/AmazonRootCA1.pem \  
--cert ~/certificate.pem.crt \  
--key ~/private.pem.key \  
--region eu-central-1 \  
--verbosity Warn \  
--max_pub_ops 1
```

Step 2: Run Sample Device Discovery

Verify Client Device Connection From Terminal

```
cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py --thing_name CMPE_Test_Client1 --topic 'clients/CMPE_Test_Client1/hello/world' --message 'Hello World!' --ca_file /home/cagatay/shared_folder/AmazonRootCA1.pem --cert /home/cagatay/shared_folder/certificate.pem.crt --key /home/cagatay/shared_folder/private.pem.key --region eu-central-1 --verbosity Warn
Performing greengrass discovery...
[WARN] [2023-11-04T15:46:21Z] [00007f5a87ffff640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice-CMPE583-GreengrassCore', cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address='10.0.2.15', metadata='', port=8883)]), certificateAuthorities=['-----BEGIN CERTIFICATE-----\nMIIDTC...CjBt\n-----END CERTIFICATE-----\n'])
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/hello/world: {"message": "Hello World!", "sequence": 0}

Publish received on topic clients/CMPE_Test_Client1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/CMPE_Test_Client1/hello/world: {"message": "Hello World!", "sequence": 1}

Publish received on topic clients/CMPE_Test_Client1/hello/world
b'{"message": "Hello World!", "sequence": 1}'
Published topic clients/CMPE_Test_Client1/hello/world: {"message": "Hello World!", "sequence": 2}

Publish received on topic clients/CMPE_Test_Client1/hello/world
b'{"message": "Hello World!", "sequence": 2}'
Published topic clients/CMPE_Test_Client1/hello/world: {"message": "Hello World!", "sequence": 3}

Publish received on topic clients/CMPE_Test_Client1/hello/world
b'{"message": "Hello World!", "sequence": 3}'
```

Step 3: Verify Client Device Connection (MQTT Bridge)

Verify From MQTT Test Client

The screenshot shows a MQTT test client interface with two main sections: "Subscribe to a topic" and "Publish to a topic".

Subscribe to a topic:

- Topic filter:** clients/+hello/world
- Info:** The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.
- Subscribe button:** An orange arrow points to this button.

Publish to a topic:

- Topic:** clients/+hello/world
- Message:** You cannot publish messages to a wildcard topic.
Please select a different topic to publish messages to.

Subscriptions:

| Topic | Action |
|----------------------|-----------|
| clients/+hello/world | Heartbeat |

Message History:

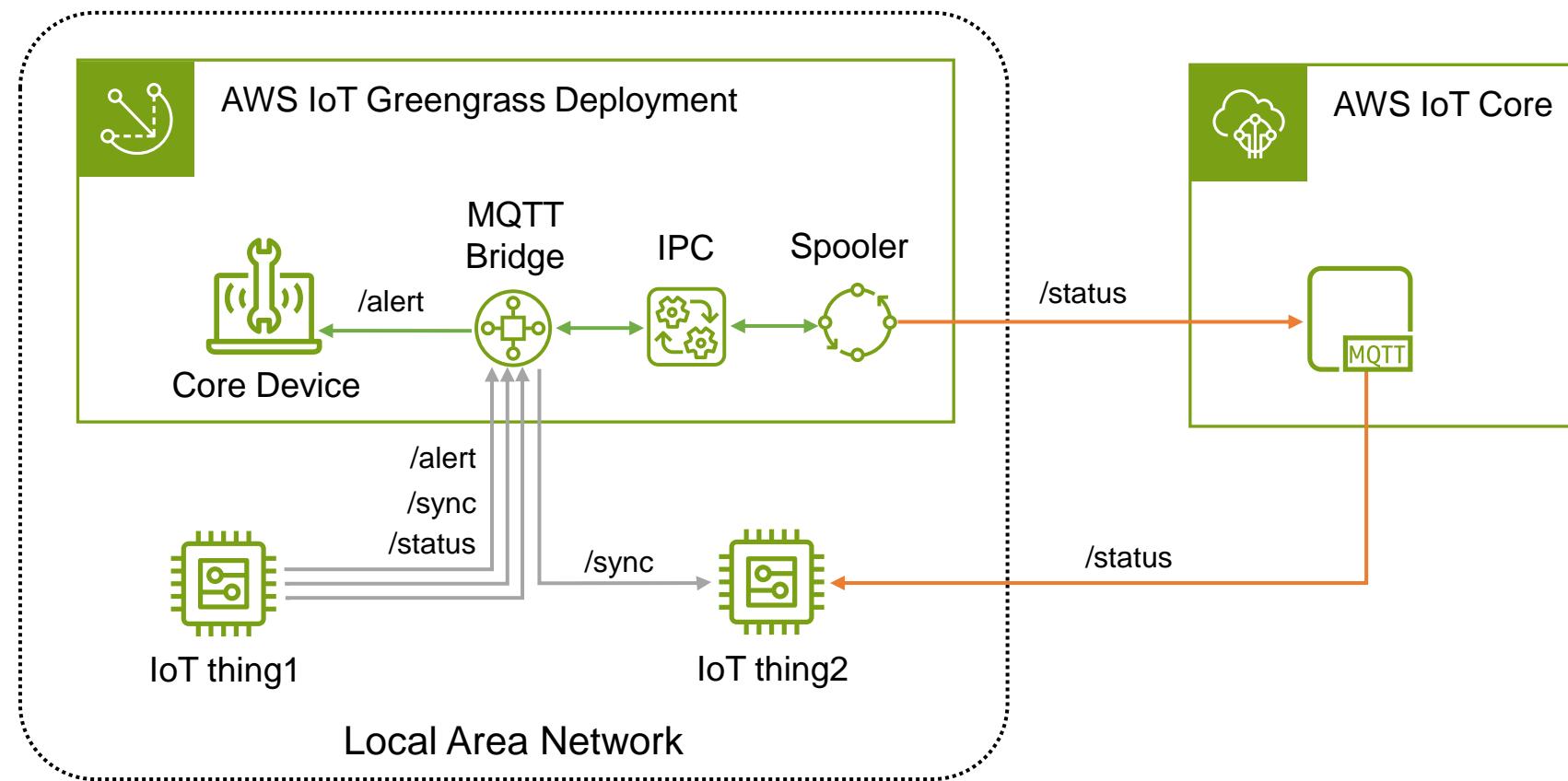
| Topic | Time |
|---------------------------------------|--|
| clients/CMPE_Test_Client1/hello/world | November 04, 2023, 18:46:25 (UTC+0300) |
| clients/CMPE_Test_Client1/hello/world | November 04, 2023, 18:46:25 (UTC+0300) |
| clients/CMPE_Test_Client1/hello/world | November 04, 2023, 18:46:24 (UTC+0300) |

Sensitivity: Internal / Non-Personal Data

Communicate with Client Devices

Communicate with Client Devices

- In this section, we will send *events* from the IoT device to the core device, the cloud (AWS IoT Core) and the other client devices through an MQTT Bridge.



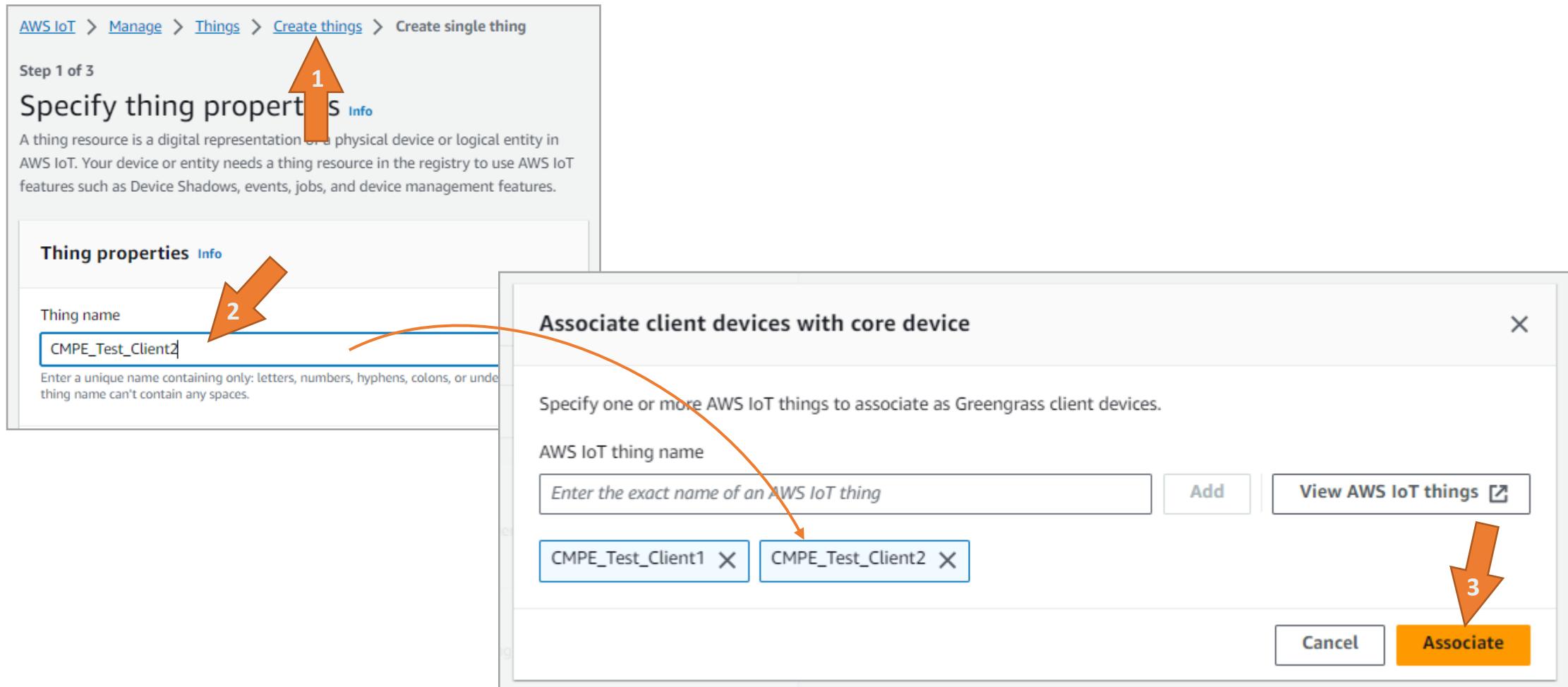
Step 1: Reconfigure Cloud Discovery

To Edit the MQTT Bridge

The screenshot shows the AWS IoT Greengrass Core device configuration interface. The left sidebar contains navigation links for Test (MQTT test client, Device Location), Manage (All devices, Things, Thing groups, Thing types, Fleet metrics), and Greengrass devices (Core devices, Components, Deployments, Groups (V1), LPWAN devices, Software packages, Remote actions, Message routing, Retained messages). A large orange arrow labeled '1' points to the 'Core devices' link. The main content area shows the device details for 'CMPE583-GreengrassCore'. The top navigation bar shows the path: AWS IoT > Greengrass > Core devices > CMPE583-GreengrassCore. An orange arrow labeled '2' points to the breadcrumb item 'Core devices'. The 'Overview' section displays the device status as 'Healthy' (indicated by a green checkmark icon) and the Greengrass Core software version as '2.11.3'. An orange arrow labeled '3' points to the 'Logs' link under the 'Status' section. Below the overview, tabs for Components, Deployments, Thing groups, Client devices (which is selected and highlighted in blue), and Tags are visible. An orange arrow labeled '4' points to the 'Configure cloud discovery' button in the 'Cloud discovery configuration' section.

Step 1: Reconfigure Cloud Discovery

Create Another Client and Associate It With Core Device



Step 1: Reconfigure Cloud Discovery

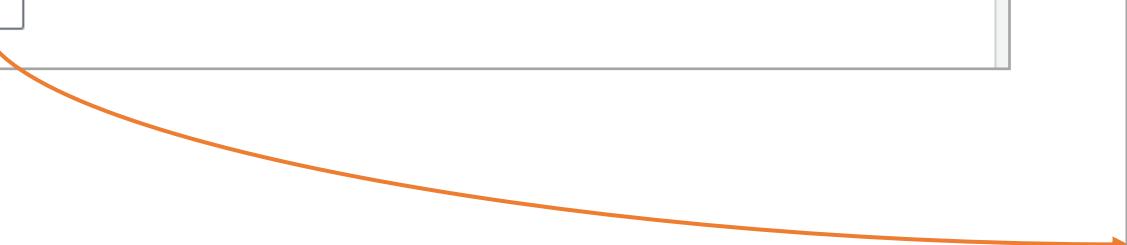
Reconfigure MQTT Bridge

Step 3: Configure and deploy Greengrass components

Greengrass core devices require a set of components to support client devices. Select and configure the components to deploy. Then, choose **Review and deploy** to open the deployment page, where you create a new deployment or revise the latest deployment for the core device target.

MQTT bridge - aws.greengrass.clientdevices.mqtt.Bridge Info
Deploy this component to relay MQTT messages between client devices, local publish/subscribe, and the AWS IoT Core cloud service. Configure this component to specify which topics to relay.

[Edit configuration](#)



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. [Learn more](#)

```
8    "ClientToClientEvents": {  
9        "topic": "clients/+sync",  
10       "source": "LocalMqtt",  
11       "target": "Pubsub"  
12    },  
13    "ClientDeviceEvents": {  
14        "topic": "clients/+alert",  
15        "targetTopicPrefix": "event/",  
16        "source": "LocalMqtt",  
17        "target": "Pubsub"  
18    },  
19    "ClientDeviceCloudStatusUpdate": {  
20        "topic": "clients/+status",  
21        "targetTopicPrefix": "update/",  
22        "source": "LocalMqtt",  
23        "target": "IotCore"  
24    }  
25 }
```

JSON Ln 26, Col 2 Errors: 0 Warnings: 0

Step 1: Reconfigure Cloud Discovery

New MQTT Topic Mapping

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. [Learn more](#)

```
8▼   "ClientToClientEvents": {  
9     "topic": "clients/+sync",  
10    "source": "LocalMqtt",  
11    "target": "Pubsub"  
12  },  
13▼  "ClientDeviceEvents": {  
14    "topic": "clients/+alert",  
15    "targetTopicPrefix": "event/",  
16    "source": "LocalMqtt",  
17    "target": "Pubsub"  
18  },  
19▼  "ClientDeviceCloudStatusUpdate": {  
20    "topic": "clients/+status",  
21    "targetTopicPrefix": "update/",  
22    "source": "LocalMqtt",  
23    "target": "IotCore"  
24  }  
25 }
```

JSON Ln 26, Col 2 ⚠ Errors: 0 ⚠ Warnings: 0

Relay messages from client devices to other clients on topics that match the clients/+sync topic filter through MQTT Bridge on Core Device.

Relay messages from client devices to local publish/subscribe on topics that match the clients/+alerts topic filter, and add the events/ prefix to the target topic. The resulting target topic matches the **events/clients/+/alert** topic filter.

Relay messages from client devices to AWS IoT Core on topics that match the clients/+status topic filter, and add the update/ prefix to the target topic. The resulting target topic matches the **update/clients/+status** topic filter.

Step 1: Reconfigure Cloud Discovery

Review & Deploy the New Configuration

MQTT bridge - aws.greengrass.clientdevices.mqtt.Bridge [Info](#)
Deploy this component to relay MQTT messages between client devices, local publish/subscribe, and the AWS IoT Core cloud service. Configure this component to specify which topics to relay.

[Edit configuration](#)

IP detector - aws.greengrass.clientdevices.IPDetector [Info](#)
Deploy this component to manage the core device's MQTT endpoints in the AWS IoT Greengrass cloud service, so client devices know where to connect. You can use this component if you have a simple network setup or client devices on the same network as the core device. Otherwise, you can manually manage the core device's endpoints.

[Edit configuration](#)

Shadow manager - aws.greengrass.ShadowManager [Info](#)
Deploy this component to enable the local shadow service, which enables you to interact with and sync client device shadows. You must configure the MQTT bridge component to relay messages between client devices and shadow manager, which uses local publish/subscribe. Client device shadow support requires Greengrass nucleus v2.6.0 or later, shadow manager v2.2.0 or later, and MQTT bridge v2.2.0 or later.

[Edit configuration](#)

[Cancel](#) [Review and deploy](#)



Step 1: Reconfigure Cloud Discovery

Check Deployment Status

Deployment successfully created

Your configuration is now being deployed. After the deployment successfully completes, you can connect client devices to the core device. To test communication, follow instructions in our documentation. [Learn more](#)

AWS IoT > Greengrass > Deployments > Deployment for CMPE583-GreengrassCore

Deployment for CMPE583-GreengrassCore

Latest revision: 1 ▾ Cancel Actions ▾

| Overview | | |
|----------------------------------|--------------------------------|------------------------------------|
| Target CMPE583-GreengrassCore | Target type Core device | Deployment created 1 minute ago |
| Device status Healthy | Deployment status Completed | |

Step 2: Create a Custom Component

To Suscribe Local MQTT Events

The screenshot shows the AWS Greengrass Components page. The navigation bar at the top includes links for AWS IoT, Greengrass, and Components. Below the navigation, the title "Greengrass components" is displayed with an "Info" link. A horizontal menu bar contains three tabs: "My components" (which is selected and underlined), "Public components", and "Community components". To the right of this menu, there is a large orange arrow pointing downwards towards the "Create component" button. The main content area is titled "My components (0)" and contains a message stating, "Your components are private components that only you can see and deploy to core devices." It includes a "Learn more" link with a blue icon. Below this is a search bar with the placeholder text "Find by name, operating system, or architecture". To the right of the search bar are navigation icons for back, forward, and settings. A table header with columns for Name, Publisher, Version, Operating systems, Architectures, and Version created is shown. The main body of the page displays the message "No components" and "You don't have any Greengrass components in us-east-1.". At the bottom center is a prominent "Create component" button.

Step 2: Create a Custom Component

Configure accessControl for Local Messages

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 {  
2     "RecipeFormatVersion": "2020-01-25",  
3     "ComponentName": "ClientDeviceEventSubscriber",  
4     "ComponentVersion": "1.0.0",  
5     "ComponentDescription": "A component that subscribes to /event messages from client devices.",  
6     "ComponentPublisher": "Amazon",  
7     "ComponentConfiguration": {  
8         "DefaultConfiguration": {  
9             "accessControl": {  
10                "aws.greengrass.ipc.pubsub": {  
11                    "ClientDeviceEventSubscriber:pubsub:1": {  
12                        "policyDescription": "Allows access to publish/subscribe to all topics.",  
13                        "operations": [  
14                            "aws.greengrass#SubscribeToTopic"  
15                        ],  
16                        "resources": [  
17                            "*"  
18                        ]  
19                    }  
20                }  
21            }  
22        }  
23    }  
24}
```

JSON Ln 53, Col 2 Errors: 0 Warnings: 0

Check GitHub to see whole contents in
cmpe583-ClientDeviceEventSubscriber.json

Allow the component to subscribe to messages from local IoT Devices.

Allow access to all topics.

Step 2: Create a Custom Component

Configure Artifacts in the Recipe

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 \[\]](#)

```
29     },
30     "Manifests": [
31         {
32             "Platform": {
33                 "os": "linux"
34             },
35             "Lifecycle": {
36                 "Install": "pip3 install --user awsiotsdk",
37                 "Run": "python3 -u {artifacts:path}/cmpe583-ClientDeviceEventSubscriber
38                     .py"
39             },
40             "Artifacts": [
41                 {
42                     "Uri": "s3://cmpe583.greengrass.bucket/artifacts/cmpe583
43                         -ClientDeviceEventSubscriber.py",
44                     "Digest": "0nsJZs7zXTg/QisJ7FWX7gneJFCSSLmTGJKZZrsF5k8=",
45                     "Algorithm": "SHA-256",
46                     "Unarchive": "NONE",
47                     "Permission": {
48                         "Read": "OwnNER",
49                         "Execute": "NONE"
50                     }
51                 }
52             ]
53         }
54     ]
55 }
```

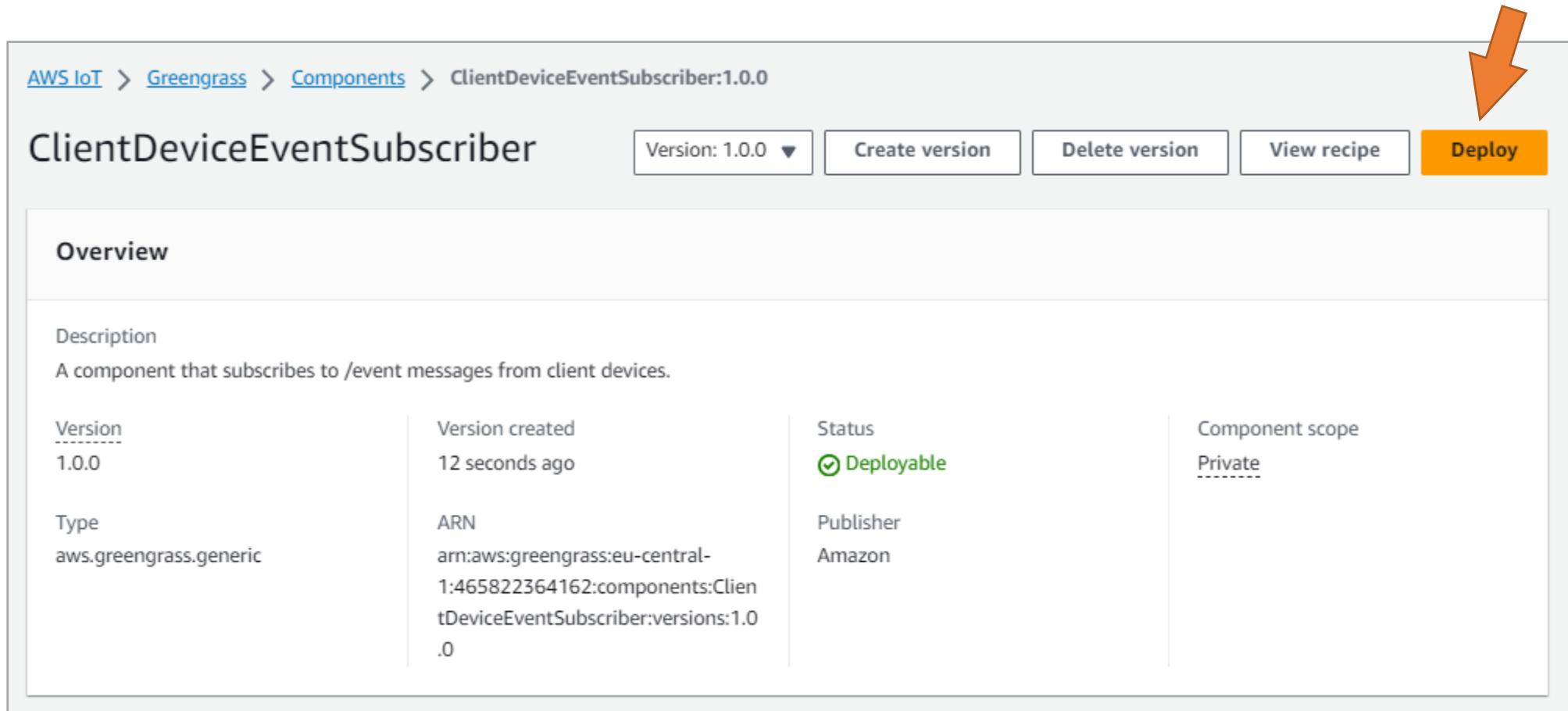
JSON Ln 53, Col 2 Errors: 0 Warnings: 0

Check GitHub to see source code in python file:
cmpe583-ClientDeviceEventSubscriber.py

Upload python file to S3 bucket and be sure that the Core Device has proper access rights to related file.

Step 3: Deploy Your Component

Via AWS IoT Greengrass (Console)



AWS IoT > Greengrass > Components > ClientDeviceEventSubscriber:1.0.0

ClientDeviceEventSubscriber

Version: 1.0.0 ▾ Create version Delete version View recipe Deploy

Overview

Description

A component that subscribes to /event messages from client devices.

| Version | Version created | Status | Component scope |
|------------------------|--|--|-----------------|
| 1.0.0 | 12 seconds ago | <input checked="" type="checkbox"/> Deployable | Private |
| Type | ARN | Publisher | |
| aws.greengrass.generic | arn:aws:greengrass:eu-central-1:465822364162:components:ClientDeviceEventSubscriber:versions:1.0.0 | Amazon | |

Step 3: Deploy Your Component

Select the Component

Select components - optional

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 (1)

| Find by name | Show only selected components | < 1 > |
|---|-------------------------------------|-------|
| <input checked="" type="checkbox"/> Name ↗ | <input checked="" type="checkbox"/> | |
| ClientDeviceEventSubscriber | | |

Public components (47)

| Find by name | Show only selected components | < 1 2 3 > |
|--|-------------------------------------|-----------|
| <input checked="" type="checkbox"/> Name ↗ | <input checked="" type="checkbox"/> | |
| aws.greengrass.Nucleus | | |
| aws.greengrass.clientdevices.Auth | | |
| aws.greengrass.clientdevices.IPDetector | | |
| aws.greengrass.clientdevices.mqtt.Bridge | | |
| aws.greengrass.clientdevices.mqtt.Moquette | | |

Cancel Skip to Review Previous Next

Step 3: Deploy Your Component

Review & Deploy

Step 4: Advanced deployment configurations

Advanced deployment configurations

Component update policy
Notify components, Component update response timeout: 60 second

Configuration validation response timeout
This deployment doesn't specify a configuration validation response timeout.

Failure handling policy
Rollback

[Download as JSON](#)

Cancel [Previous](#) **Deploy**



Step 3: Deploy Your Component

Check Deployment Status on AWS IoT Console

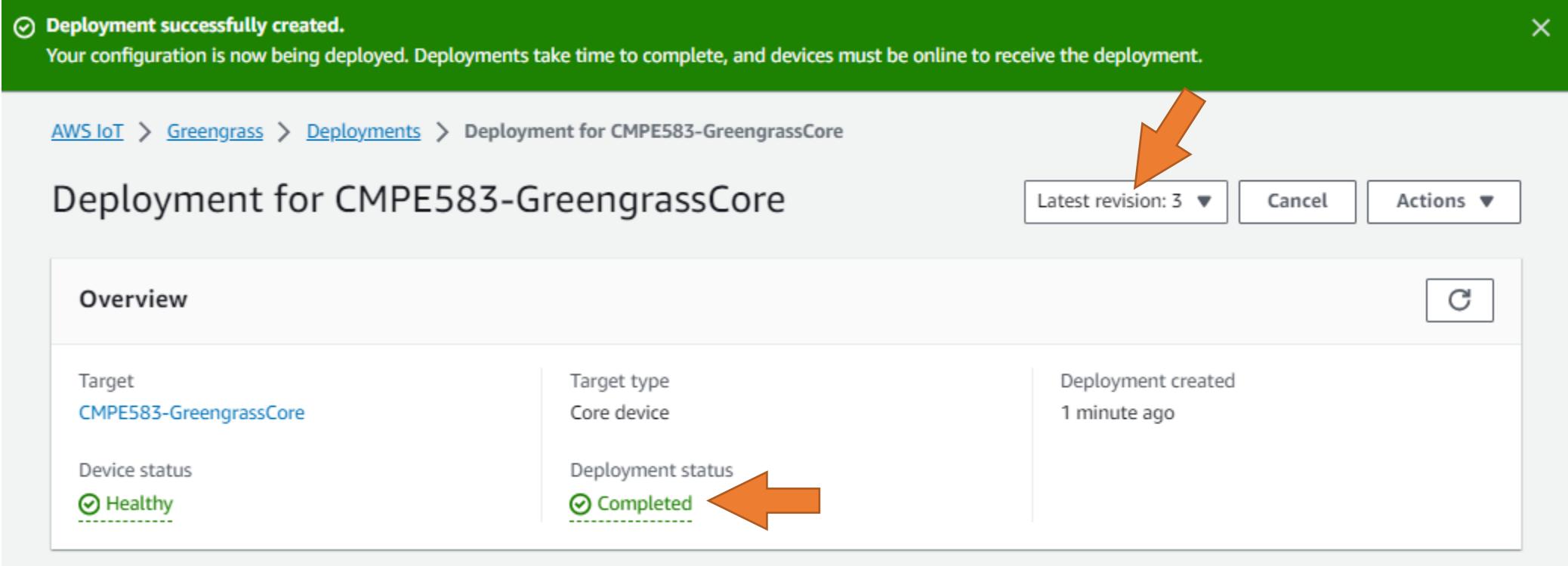
Deployment successfully created.
Your configuration is now being deployed. Deployments take time to complete, and devices must be online to receive the deployment.

AWS IoT > Greengrass > Deployments > Deployment for CMPE583-GreengrassCore

Deployment for CMPE583-GreengrassCore

Latest revision: 3 ▾ Cancel Actions ▾

| Overview | C | |
|---|--|------------------------------------|
| Target CMPE583-GreengrassCore | Target type Core device | Deployment created 1 minute ago |
| Device status 健康的 | Deployment status 完成 | |



Step 3: Deploy Your Component

Check Deployment Status on Core Device

- Check ClientDeviceEventSubscriber.log file if the deployed component runs without error.
- If yes, you will see that the Core Device was subscribed to the topic successfully.

```
root@kubuntu:/greengrass/v2/logs# tail -f ClientDeviceEventSubscriber.log
2023-11-05T13:56:07.780Z [INFO] (pool-2-thread-40) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW, command=["pip3 install --user awsiotsdk"]}
2023-11-05T13:56:08.473Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: awsiotsdk in /usr/local/lib/python3.10/dist-packages (1.19.0). {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2023-11-05T13:56:08.476Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: awscrt==0.19.1 in /usr/local/lib/python3.10/dist-packages (from awsiotsdk) (0.19.1). {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2023-11-05T13:56:08.608Z [INFO] (pool-2-thread-40) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=STARTING, command=["python3 -u /greengrass/v2/packages/artifacts/ClientDeviceEventSubscriber/1.0.0..."]}
2023-11-05T13:56:08.697Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Successfully subscribed to topic: event/clients/+/_alert. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=RUNNING}
```

Step 4: Publish 'clients+/alert' Topic From IoT Client Publish Topic to Trigger Subscribers (Core Device)

- Publish the topic with basic_discovery.py script.

```
(venvclient)$ cd aws-iot-device-sdk-python-v2/samples
```

```
(venvclient)$ python3 basic_discovery.py \
--thing_name CMPE_Test_Client1 \
--topic 'clients/CMPE_Test_Client1/alert' \
--message 'Production Anomaly Detected. Check the Production Plant!' \
--ca_file ~/AmazonRootCA1.pem \
--cert ~/certificate.pem.crt \
--key ~/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1 \
--mode publish
```

This message will be relayed to Core Device

```
"ClientDeviceEvents": [
    "topic": "clients+/alert",
    "targetTopicPrefix": "event/",
    "source": "LocalMqtt",
    "target": "Pubsub"
},
```

Step 4: Publish 'clients/+/alert' Topic From IoT Client Verify Process on Terminal

```
cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
--thing_name CMPE_Test_Client1 \
--topic 'clients/CMPE_Test_Client1/alert' \
--message 'Production Anomaly Detected. Check the Production Plant!' \
--ca_file /home/cagatay/shared_folder/AmazonRootCA1.pem \
--cert /home/cagatay/shared_folder/certificate.pem.crt \
--key /home/cagatay/shared_folder/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1
Performing greengrass discovery...
[WARN] [2023-11-05T08:38:23Z] [00007f368d0fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice-CMPE583-GreengrassCore', cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address='10.0.2.15', metadata='', port=8883)]), certificateAuthorities=['-----BEGIN CERTIFICATE-----\nMIID1TCCAr2gAwIBAgIVAMDxNyNBeW2Jja8e093NSaPTIV8WMA0GCSqGSIb3DQE\nBnCwUAMIGJMQswCQYDVQ\nQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBJbmMuMRww\\nGgYDVQLDBNbWF6b24gV2ViIFNlc\nzP\nY2VzMRMwEQYDVQQIDApXYXNoaW5ndG9u\\nMRAwDgYDVQQHAdTZWF0dG\nxLMRswGQYDVQQDBJHcmVlbmdyYXNzIE\nNvcnUgQ0Ew\\nHhcNMjMxMTA0MTMwMTM5WhcNMjgxMTAyMTMwMTM5WjCBiTELMAkGA1UEBhMCVVMx\\nGDAwBgNVBAoMD0FtYXpbvi5jb2\n0gSw5jLjEcMBoGA1UECwwTQW1hem9uIFd1YiBT\\nZXJ2aWNlc\nzETMBEGA1UECAwKV2FzaGluZ3RvbjEQMA4GA1UEBwwHU2VhdHRsZTEb\\nMBkGA1UEAw\nSR3JlZW5ncmFzcyBD\nb3JLIENB\nMIIBIjANB\nbgkqhkiG9w0BAQEFAAO\nC\\nAQ8AMII\nBCgKCAQEAmZ2lfIUSehjcUmSgkm1uCt8VoKcQtWIoy0pzVmyNaR2RnDFr\\n48huzRAT+Ee6dVxyLMRcs\nw1ncJM62cPv8w\nATNDgC/IMdtotLLqwAGKdcFfF\niWvt\nx\\nEZ1zfN8jsQj8Mutr4X+PfFmFSqxSTdfTXB2l7Z0SX0SE7auaLzHcH8HTDYAklp2h\\nXeK4VMxiKbHne4KdoPbQgk8m83BniA0WOKOTab\n9wFjwi+1wd875jV3r1SS64uz+9\\ncJ6Pp+4Kv+cvjZyMWPTnSKI4QoX1bQd+921XZwqtGCTvzAPdLm5E/B8ber2/lb2s\\nwJ4f8A7jeQs5VtDz\nmVoeXld8Sdm/ocji4+pGqQIDAQ\nABozIwMDAPBgNVHRMBAf8E\\nBTADAQH/MB0GA1UdDgQWBBQyiqUi\npUYO\nXyyDknCF/imtYlYyLzANB\nbgkqhkiG9w0B\\nAQsFAA0CAQEA\nYfoi\nBX5fNJRpm/MW1VVdE62f7K+30atTP\noroZa6CTgUL\nhdTnWG\\nWP5MIZt9iKq3AQzgma7aQEt/dKJ\nJL3nmDqsY/fHRjjjcF6SSu1W4Fp32BFp3mi57\\naeICvFqMgHztyBB5WJVsU7EktWSn0PjWSgcB\nWticXFw8a0G55H\nS5rrUqi9mBYWSm\\njo/7rVsX/U0kUHP19sBu6JrJarvibkUdyxY66m0EpLMmf9My/11TIL5s2rh\npqRSX\\nR9DsiyGAq7KXaJzMsxWn1DxvlyRfjNo3R3W0ICWq1F6y6RtwU+ofAo\nTxMbC25XQ0\\ncYI/qnKkj0/K4ycY0QPZNTqr76I5INP5iA==\\n-----END CERTIFICATE-----\\n')])\nTrying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883\nConnected!\nPublished topic clients/CMPE_Test_Client1/alert: {"message": "Production Anomaly Detected. Check the Production Plant!", "sequence": 0}\n\nPublish received on topic clients/CMPE_Test_Client1/alert\nb'{"message": "Production Anomaly Detected. Check the Production Plant!", "sequence": 0}'
```

Step 4: Publish 'clients/+/alert' Topic From IoT Client

Verify Process on Core Device's Terminal

```
root@kubuntu:/greengrass/v2/logs# tail -f ClientDeviceEventSubscriber.log
2023-11-04T22:06:57.422Z [INFO] (pool-2-thread-168) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW, command=["pip3 install --user awsiotsdk"]}
2023-11-04T22:06:57.837Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: aws iotsdk in /usr/local/lib/python3.10/dist-packages (1.19.0). {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2023-11-04T22:06:57.840Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: aws crt==0.19.1 in /usr/local/lib/python3.10/dist-packages (from awsiotsdk) (0.19.1). {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2023-11-04T22:06:57.954Z [INFO] (pool-2-thread-168) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=STARTING, command=["python3 -u /greengrass/v2/packages/artifacts/ClientDeviceEventSubscriber/1.0..."]}
2023-11-04T22:06:58.025Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Successfully subscribed to topic: clients/+/event/alert. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=RUNNING}

2023-11-05T08:28:24.625Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Received new event from IoT Client : {"message": "Production Anomaly Detected. Check the Production Plant!", "sequence": 0}. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=RUNNING}
```

Step 5: Publish 'clients+/status' Topic From IoT Client Publish Topic to Trigger Subscribers (AWS IoT Core)

- Publish the topic with basic_discovery.py script.

```
(venvclient)$ cd aws-iot-device-sdk-python-v2/samples  
  
(venvclient)$ python3 basic_discovery.py \  
--thing_name CMPE_Test_Client1 \  
--topic 'clients/CMPE_Test_Client1/status' \  
--message 'Client device is running for 2 days without error!' \  
--ca_file ~/AmazonRootCA1.pem \  
--cert ~/certificate.pem.crt \  
--key ~/private.pem.key \  
--region eu-central-1 \  
--verbosity Warn \  
--max_pub_ops 1 \  
--mode publish
```

This message will be relayed
to AWS IoT Core

```
"ClientDeviceCloudStatusUpdate": {  
    "topic": "clients+/status",  
    "targetTopicPrefix": "update/",  
    "source": "LocalMqtt",  
    "target": "IotCore"  
}
```

Step 5: Publish 'clients+/status' Topic From IoT Client Verify Process on Terminal

```
cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
--thing_name CMPE_Test_Client1 \
--topic 'clients/CMPE_Test_Client1/status' \
--message 'Client device is running for 2 days without error!' \
--ca_file /home/cagatay/shared_folder/AmazonRootCA1.pem \
--cert /home/cagatay/shared_folder/certificate.pem.crt \
--key /home/cagatay/shared_folder/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1
Performing greengrass discovery...
[WARN] [2023-11-05T08:41:03Z] [00007f61829fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice-CMPE583-GreengrassCore', cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address='10.0.2.15', metadata='', port=8883)]), certificateAuthorities=['-----BEGIN CERTIFICATE-----\nMIID1TCCAr2gAwIBAgIVAMDxNyNBeW2Jja8e093NSaPTIV8WMA0GCSqGSIb3DQEBAQc\nnCwUAMIGJMQswCQYDVQ\nQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBjbmMuMRw\nnGgYD\nVQLDBNbWF6b24gV2ViIFNlc\nZpY2VzMRMwEQYDVQ\nQIDApXYXNoaW5ndG9u\nnMRAwDgYD\nVQHAdTZWF0dG\nx1MRswGQYDVQ\nQDDJBHcmVlbmdyYXNzIE\nNvcnUgQ0Ew\nnHcNMjMxMTA0MTMwMTM5WhcNMjgxMTAyMTMwMTM5WjCBiTELMAkGA1UEBhMCVV\nMx\nnGDAwBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMB0GA1UECw\nTQW1hem9uIFd1YiBT\nnZXJ2awNlc\nzETMBEGA1UECAwKV2FzaGluZ3RvbjEQMA4GA1UEBwwHU2VhdHRsZTEb\nnMBkGA1UEAw\nSR3JLZW5ncmFzcyBD\n3J1IENBMII\nB1jANBkgkhkiG9w0BAQEF\nAAOC\nnAQ8AM\nIIBCgKCAQEAmZ2lfIUSehjcUmSgkm1uCt8VoKcQtWIoy0pzVmyNaR2RnDFr\nn48huzRAT+Ee6dVxyLMRcs\nw1ncJM62cPv8wATNDgC/IMdtotLLqwAGKdcFfFiWvt\nnEZ1zfN8JsQJ8Mutr4X+PfFmFSqxStdfTXB2l7Z0SX0SE7auaLzHcH8HTDYAklp2h\nnXeK4VMxiKbHne4KdoPbQgk8m83BniA0W0K0Tab9wFjwi+1wd875jV3r1SS64uz+9\nncJ6Pp+4Kv+cvjZyMWPTnSKI4QoXlbQd+92lXZwqtGCTvzAPdLm5E/B8ber2/lb2s\nnwJ4f8A7jeQs5VtDZmVoeXLd8Sdm/ocji4+pGqQIDAQABozIwMDAPBgnVHRMBAf8E\nnBTADAQH/MB0GA1UdDgQWB\nBQyiqUi\npUYO\nXyyDknCF/imtYlYyLzANBkgkhkiG9w0B\nnAQSFAAO\nCAQEAYfoiiBX5fNJRpm/MW1VVdE62f7K+30atTPoroZa6CTgULZhdTnWG\nnWP5MIZt9iKq3AQzgma7aQEt/dKJJL3nmDqsY/fHRjjcF6SSu1W4Fp32BFp3mi57\\naeICvFqMgHztyB\nB5WJVsU7EktWSn0PjWSgcB\nWticXFw8a0G55HS5rrUqi9mBYWSm\nnjo/7rVsX/U0kUHP19sBu6JrJarvibkUdyxY66m0EpLMmf9My/11TIL5s2rhpqRSX\nnR9Ds\niyGAq7KXaJzMsxWn1DxvlyRfjNo3R3W0ICWq1F6y6RtwU+ofAoTxMbC25XQ0\\ncYI/qnKkj0/K4ycY0QPZN7qr76I5INP5iA==\\n-----END CERTIFICATE-----\\n'])])
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/status: {"message": "Client device is running for 2 days without error!", "sequence": 0}
Published topic clients/CMPE_Test_Client1/status: {"message": "Client device is running for 2 days without error!", "sequence": 0}
Publish received on topic clients/CMPE_Test_Client1/status
b'{"message": "Client device is running for 2 days without error!", "sequence": 0}'
```

Step 5: Publish 'clients+/status' Topic From IoT Client

Verify Process on MQTT Test Client

The screenshot shows an MQTT test client interface with two main sections: 'Subscribe to a topic' and 'Publish to a topic'. The 'Subscribe to a topic' section is active, featuring a 'Topic filter' input field containing 'update/clients+/status'. An orange arrow points to this input field. Below it is a 'Subscribe' button. The 'Publish to a topic' section is inactive. In the bottom half of the interface, under 'Subscriptions', there is a list with one item: 'update/clients+/status'. To the right of this list are four buttons: 'Pause', 'Clear', 'Export', and 'Edit'. Below the subscription list, a detailed view of a received message is shown. The message is from the topic 'update/clients/CMPE_Test_Client1/status' and was received on 'November 05, 2023, 11:43:37 (UTC+0300)'. The message content is a JSON object: { "message": "Client device is running for 2 days without error!", "sequence": 0 }. This message content is highlighted with a red dotted rectangle.

Subscribe to a topic Publish to a topic

Topic filter | Info
The topic filter describes the topics which you want to subscribe. The topic filter can include MQTT wildcard characters.

update/clients+/status

► Additional configuration

Subscribe

Subscriptions update/clients+/status

update/clients+/status

Pause Clear Export Edit

▼ update/clients/CMPE_Test_Client1/status November 05, 2023, 11:43:37 (UTC+0300)

```
{  
  "message": "Client device is running for 2 days without error!",  
  "sequence": 0  
}
```

Step 6: Publish 'clients/+/sync' Topic From IoT Client Subscribe Topic From Test Client 2

- Subscribe the topic using the '*subscribe*' --mode argument of basic_discovery.py script.

```
python3 -m venv venvclient2
source venvclient2/bin/activate
(venvclient2)$ python3 -m pip install ./aws-iot-device-sdk-python-v2
(venvclient2)$ cd aws-iot-device-sdk-python-v2/samples

(venvclient2)$ python3 basic_discovery.py \
--thing_name 'CMPE_Test_Client2' \
--topic 'clients/+/sync' \
--ca_file ~/AmazonRootCA1.pem \
--cert ~/certificate.pem.crt \
--key ~/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--mode subscribe
```

Step 6: Publish 'clients/+/sync' Topic From IoT Client Publish Topic to Trigger Subscribers (Test Client 2)

- Publish the topic using the '*publish*' --mode argument of basic_discovery.py script.

```
(venvclient)$ cd aws-iot-device-sdk-python-v2/samples
```

```
(venvclient)$ python3 basic_discovery.py \
--thing_name 'CMPE_Test_Client1' \
--topic 'clients/CMPE_Test_Client1/sync' \
--message 'Hello from IoT Client 1!' \
--ca_file ~/AmazonRootCA1.pem \
--cert ~/certificate.pem.crt \
--key ~/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1 \
--mode publish
```

This message will be relayed
to Core Device (MQTT Bridge)

```
"ClientToClientEvents": [
    "topic": "clients/+/sync",
    "targetTopicPrefix": "local/",
    "source": "LocalMqtt",
    "target": "Pubsub"
],
```

Step 6: Publish 'clients/+/sync' Topic From IoT Client

Verify Topic Published

```
(venvclient1) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
--thing_name 'CMPE_Test_Client1' \
--topic 'clients/CMPE_Test_Client1/sync' \
--message 'Hello from IoT Client 1!' \
--ca_file ~/shared_folder/AmazonRootCA1.pem \
--cert ~/shared_folder/certificate.pem.crt \
--key ~/shared_folder/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1 \
--mode publish
Performing greengrass discovery...
[WARN] [2023-11-06T13:52:20Z] [00007fd3ac8fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice22364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address='AwIBAgIVAJ55wdMBGrA7mIHUCs183HgzuqAnMA0GCSqGSIb3DQEB\ncwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBjbmMuMRww\nGgYDVQDNzIENvcUgQ0Ew\nnHcNMjMxMTA2MTE1ODE0WhcNMjgxMTA0MTE1ODE0WjCBiTELMAkGA1UEBhMCVVMx\nGDAWBgNVBAoMD0FtYXpbvi5jb20gSW5jLjEcMB0GA1UE0FzcyBDb3JlIENBmIIBIjANBgkqhkiG9w0BAQEFAAOAcnAQ8AMIIIBCgKCAQEAsfbYoDMM56kG2MLGqo3/6RUFDquIN97/4qQ/CD727F45WHPd\nnuwbBtACsSXXSvc7F3k1Z3\nnwhEobHrDR8hFmjBQ5y1u0Wj3mIpScCELAtudjScHHQV0dbcium+f0vn+u0hIV0x0\nnYydkscLyVyPYcL0h7D7B0n1hsUv3WT0gYcmh1zSN49/29D9UdG07i01rWQrz9sINS7QZLNS1zANBgkqhkiG9w0B\nnAQSFAA0CAQEAMd5Pmw42K0ZWvV9ZAM1W3g2D4egNa98jBogP8otc4d0g3vhtFtVF\nn+jWWF5kinHsMVv0uQTjxV6Rf17PiYCRts3xobU47Xo5w2nl4Z+IuCJGeT3LKm1arGR4is4c2SpImAdUkjEU8SGh\nnHEkifLNFejNBx/wurqPyPRk8yw6R9c1j7ExHFHlYRLpzBguUfcy+pNDUiP E6A4jHp
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/sync: {"message": "Hello from IoT Client 1!", "sequence": 0}
```

Step 6: Publish 'clients/+/sync' Topic From IoT Client

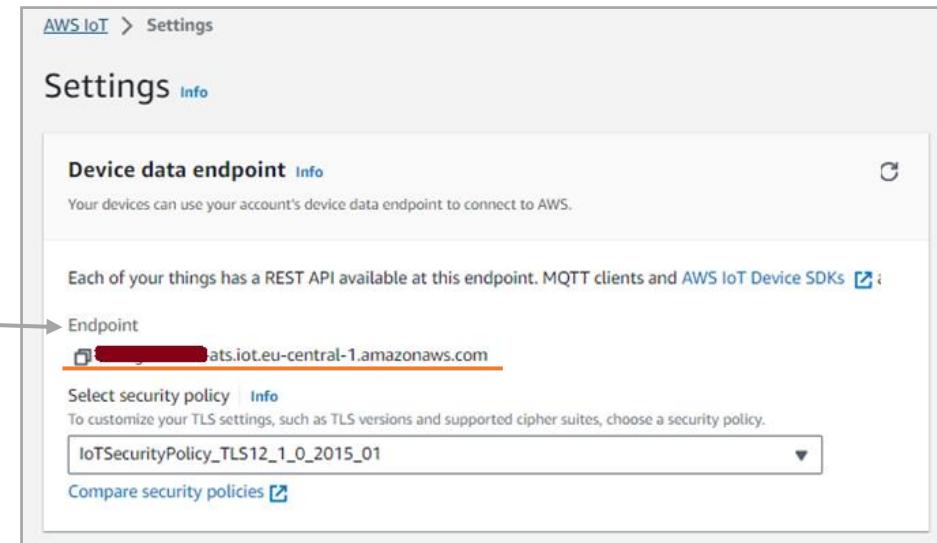
Verify Topic Received

```
(venvclient2) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
--thing_name 'CMPE_Test_Client2' \
--topic 'clients/+/sync' \
--ca_file ~/shared_folder/AmazonRootCA1.pem \
--cert ~/shared_folder/client2-certificate.pem.crt \
--key ~/shared_folder/client2-private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1 \
--mode subscribe
Performing greengrass discovery...
[WARN] [2023-11-06T13:51:22Z] [00007fd0277fe640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDev
22364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_addre
AwIBAgIVAJ5S5wdMBGrA7mIHUCs183HgzuqAnMA0GCSqGSIb3DQEByCwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBjbmMuMRww\nGgYD
NzIEvcmUgQ0Ew\nNhcmNMjMxMTA2MTE1ODE0WhcNMjgxMTA0MTE1ODE0WjCBiTELMAkGA1UEBhMCVVMx\nnGDAWBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMBoGA1U
FzcyBDb3JlIENBMIIBjANBgkqhkiG9w0BAQEFAAOC\nnAQ8AMIIIBCgKCAQEAsfbYoDMM56kG2MLGqo3/6RUFDquIN97/4qQ/CD727F45WHPd\nnuwbBtACsSXXSvc
k1Z3\nwhEobHrDR8hFmjBQ5y1u0Wj3mIpScCELAtudjSChHQV0dbcium+f0vn+u0hIV0x0\nnYydkscLyVyPYcL0h7D7B0n1hsUv3WT0gYcmh1zSN49/29D9UdG07
rWQrz9sINST7QZLNS1zANBgkqhkiG9w0B\nnAQsFAA0CAQEAMd5Pmw42K0ZWvV9ZAM1W3g2D4egNa98jBogP8otc4d0g3vhtFtVF\nn+jWWF5kinHsMVvOuQTjxV6Rf
iYCRts3xobU47Xo5w2nl4Z+IuCJGeT3Lkm1arGR4is4c2SpImAdUkjEU8SGh\nnHEkifLNFejNBx/wurqPyPRk8yw6R9c1j7ExHFhLYRLpzBgfUfcy+pNDUi
e6A4j
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Publish received on topic clients/CMPE_Test_Client1/sync
b'{"message": "Hello from IoT Client 1!", "sequence": 0}'
```

Receiving Events From AWS IoT Core

- Subscribe events published from AWS IoT Cloud with pubsub.py script using an empty --message.
 - Don't forget to modify MQTT Bridge for topic mapping!
-
- ✓ Use an empty--message with pubsub.py script to subscribe and receive events on IoT client device.
 - ✓ Your account's endpoint can be found under the settings page of AWS IoT Console

```
(venvclient)$ cd aws-iot-device-sdk-python-v2/samples  
  
(venvclient)$ python3 pubsub.py \  
  --topic 'topic-name' \  
  --message '' \  
  --endpoint 'your-endpoint' \  
  --ca_file ~/AmazonRootCA1.pem \  
  --cert ~/certificate.pem.crt \  
  --key ~/private.pem.key \  
  --region eu-central-1
```



Step 7: Publish 'clients/+/update' Topic From IoT Client Subscribe Topic From Test Client 2

- Subscribe **IoT Core events** using empty --message argument with pubsub.py script.

```
(venvclient2)$ cd aws-iot-device-sdk-python-v2/samples
```

```
(venvclient2)$ python3 pubsub.py \
    --topic 'update/clients/+/status' \
    --message '' \
    --endpoint 'your-endpoint' \
    --ca_file ~/AmazonRootCA1.pem \
    --cert ~/certificate.pem.crt \
    --key ~/private.pem.key
```

Step 7: Publish 'clients/+/update' Topic From IoT Client

Publish Topic to Trigger Subscribers(AWS Iot Core -> IoT Client2)

- Publish the topic using the '*publish*' --mode argument of basic_discovery.py script.

```
(venvclient)$ cd aws-iot-device-sdk-python-v2/samples
```

```
(venvclient)$ python3 basic_discovery.py \
--thing_name CMPE_Test_Client1 \
--topic 'clients/CMPE_Test_Client1/status' \
--message 'Status update from Client 1!' \
--ca_file ~/AmazonRootCA1.pem \
--cert ~/certificate.pem.crt \
--key ~/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1 \
--mode publish
```

This message will be relayed
to AWS IoT Core

```
"ClientDeviceCloudStatusUpdate": {
    "topic": "clients/+/status",
    "targetTopicPrefix": "update/",
    "source": "LocalMqtt",
    "target": "IoTCore"
}
```

Step 7: Publish 'clients/+/sync' Topic From IoT Client

Verify Topic Published

```
(venvclient1) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
    --thing_name 'CMPE_Test_Client1' \
    --topic 'clients/CMPE_Test_Client1/status' \
    --message 'Status update from Client 1!' \
    --ca_file ~/shared_folder/AmazonRootCA1.pem \
    --cert ~/shared_folder/certificate.pem.crt \
    --key ~/shared_folder/private.pem.key \
    --region eu-central-1 \
    --verbosity Warn \
    --max_pub_ops 1 \
    --mode publish
Performing greengrass discovery...
[WARN] [2023-11-06T15:00:26Z] [00007f511e1fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDev
22364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_
addr='AwIBAgIVAJ55wdMBGrA7mIHUCs183HgzuqAnMA0GCSqGSIb3DQEBy\nCwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBjbmMuMRww\nGgYDV
NzIEvcmUgQ0Ew\nHhcNMjMxMTA2MTE1ODE0WhcNMjgxMTA0MTE1ODE0WjCBiTELMAkGA1UEBhMCVVMx\nGDAWBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMB0GA1U
FzcyBDb3JlIENBMMIBIjANBgkqhkiG9w0BAQEFAAOCAQKCAQEAsfbYoDMM56kG2MLGqo3/6RUFdquIN97/4qQ/CD727F45WHPd\nnuwbBtACsSXXSv
k1Z3\nwhEobHrDR8hFmjBQ5y1u0Wj3mIpScCELAtudjScHHQV0dbcium+f0vn+u0hIV0x0\nYydkscLyVpYcL0h7D7B0n1hsUv3WT0gYcmh1zSN49/29D9UdG07i
rWQrz9sINS7QZLNS1zANBgkqhkiG9w0B\nAQsFAAOCQAQEMd5Pmw42K0ZWvV9ZAM1W3g2D4egNa98jBogP8otc4d0g3vhtFtVF\n+jWWF5kinHsMVv0uQTjxV6Rf1
iYCRts3xobU47Xo5w2nl4Z+IuCJGeT3LKm1arGR4is4c2SpImAdUkjEU8SGh\nHEkifLNFejNBx/wurqPyPRk8yw6R9c1j7ExHFHlYRLpzBgUfcy+pNDUi
e6A4jh
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/status: {"message": "Status update from Client 1!", "sequence": 0}
```

Step 7: Publish 'clients/+/sync' Topic From IoT Client

Verify Topic Received

```
(venvclient2) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 pubsub.py \
    --topic 'update/clients/+/status' \
    --message '' \
    --endpoint '[REDACTED]-ats.iot.eu-central-1.amazonaws.com' \
    --ca_file ~/shared_folder/AmazonRootCA1.pem \
    --cert ~/shared_folder/client2-certificate.pem.crt \
    --key ~/shared_folder/client2-private.pem.key
Connecting to [REDACTED].iot.eu-central-1.amazonaws.com with client ID 'test-43601056-a33e-428e-a538-e4be07b9aff6'...
Connection Successful with return code: 0 session present: False
Connected!
Subscribing to topic 'update/clients/+/status'...
Subscribed with QoS.AT_LEAST_ONCE
Waiting for all messages to be received...
Received message from topic 'update/clients/CMPE_Test_Client1/status': b'{"message": "Status update from Client 1!", "sequence": 0}'
```

Thank You

