



# Edge Computing with AWS IoT Greengrass - Part #1

by Cagatay Sonmez, 31 Nov 2025

# 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 repository:
  - <https://github.com/CagataySonmez/AWS-IoT-Greengrass-Demo>

# Prerequisite: Amazon Web Services (AWS)

- **What is AWS?**
  - It is a cloud platform built on the "**XaaS**" (Everything as a Service) philosophy.
  - Think of AWS as a massive, **on-demand toolbox** of digital services—like storage, servers, and security—that you can access over the internet.
  - We **don't need** to buy or build our own data centers.
- **Why Are We Using It Today?**
  - In our project, AWS acts as the "**Command Center**" or "**Brain**" for our smart device.
  - We use it to:
    - **Deploy** our code.
    - **Manage** our device from anywhere.
    - **Communicate** with our device securely .

# Our Core Services in this Lecture

## (Identity & Storage)

### IAM (Identity & Access Management)

- **What it is:** The "Security Guard" for your AWS account. It manages *who* (Users, Devices) can access what (Services).
- **How we use it:** To create *Roles* (an "identity" for our device) and *Policies* (a "permission slip" in JSON format) that allow our device to securely talk to the cloud .

### S3 (Simple Storage Service)

- **What it is:** A giant "Hard Drive in the Cloud." You store your files in folders called "*Buckets*".
- **How we use it:** We store our Python code (called *Artifacts*) in an S3 Bucket . Greengrass then automatically downloads this code from S3 to the device .

# Our Core Services in this Lecture (Compute & Messaging)

## AWS Lambda

- **What it is:** A "serverless" service. It lets you run code without thinking about servers. You just upload your script, and AWS runs it for you.
- **How we use it:** We'll create a Lambda function in the cloud and then show how Greengrass can *import* it and run that same function locally on our edge device .

## AWS IoT Core

- **What it is:** The "Post Office" or "Message Hub" for all your IoT devices. It's a secure *MQTT Broker* that handles millions of messages.
- **How we use it:** To send messages *from* the cloud to our device (e.g., 'cmpe/request') and receive responses back . We also use its *MQTT Test Client* to check our work .

---

## Key Concept to Remember:

- **Region:** A physical AWS data center location (e.g., eu-central-1 in Frankfurt).
- Greengrass components (S3 Bucket, IoT Thing, Lambda) must be in the same region!

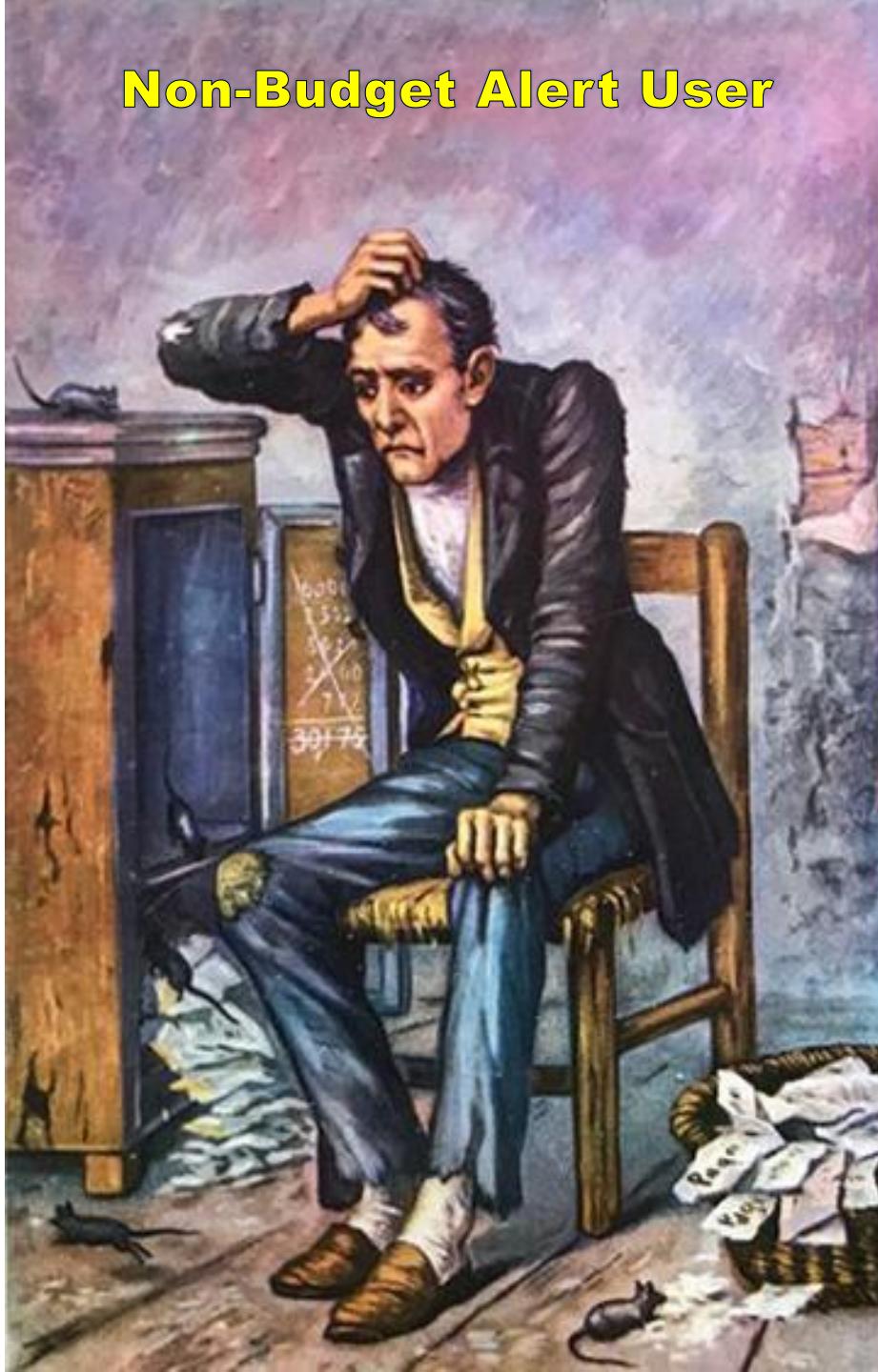
# ⚠️ Reminder

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**

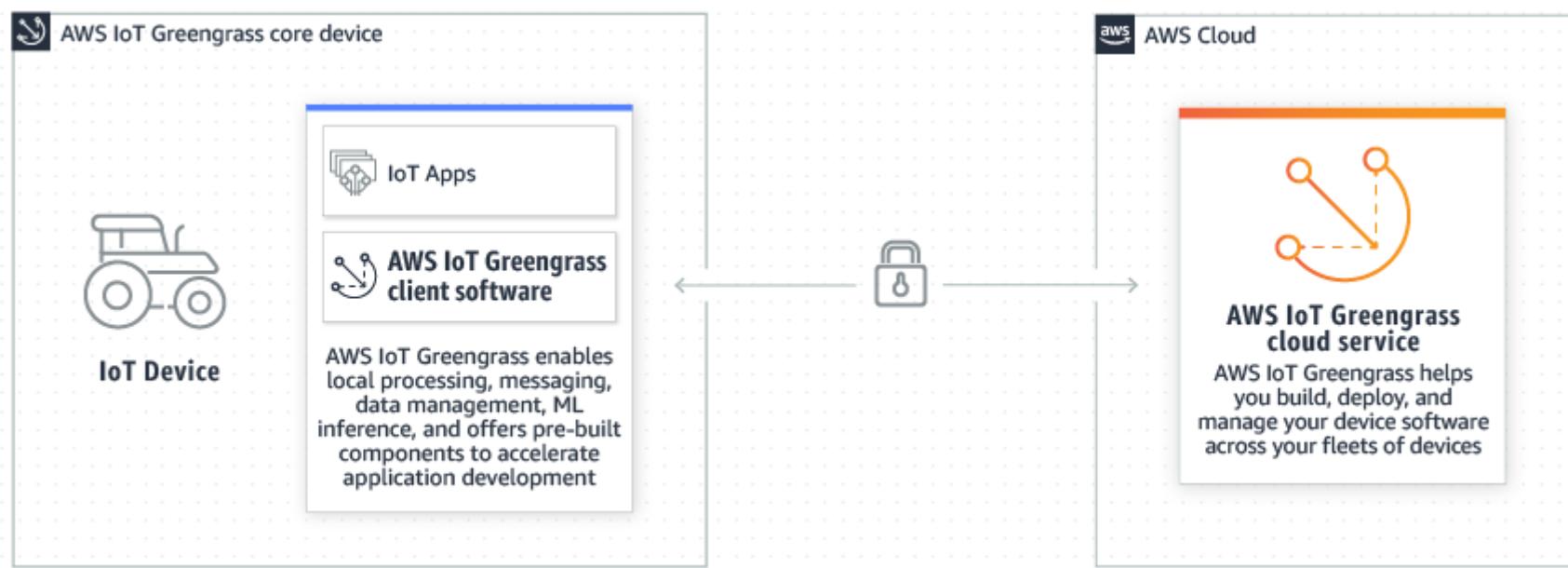


**Budget Alert User**



# 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 Cloud<sup>1</sup>.



<sup>1</sup> <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.



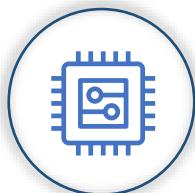
**Greengrass Core (Client) Software:** The set of all AWS IoT Greengrass software that you install on a core device. AWS IoT Greengrass Core software comprises the Nucleus and other components. It is provided by AWS that enables a device to connect to AWS IoT Core.



**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.



**Greengrass Client Device:** Any device (such as an IoT device, mobile app, or system) that communicates with a Greengrass Core device locally or via AWS IoT Core over MQTT, using the AWS IoT Device SDK.



Physical Device

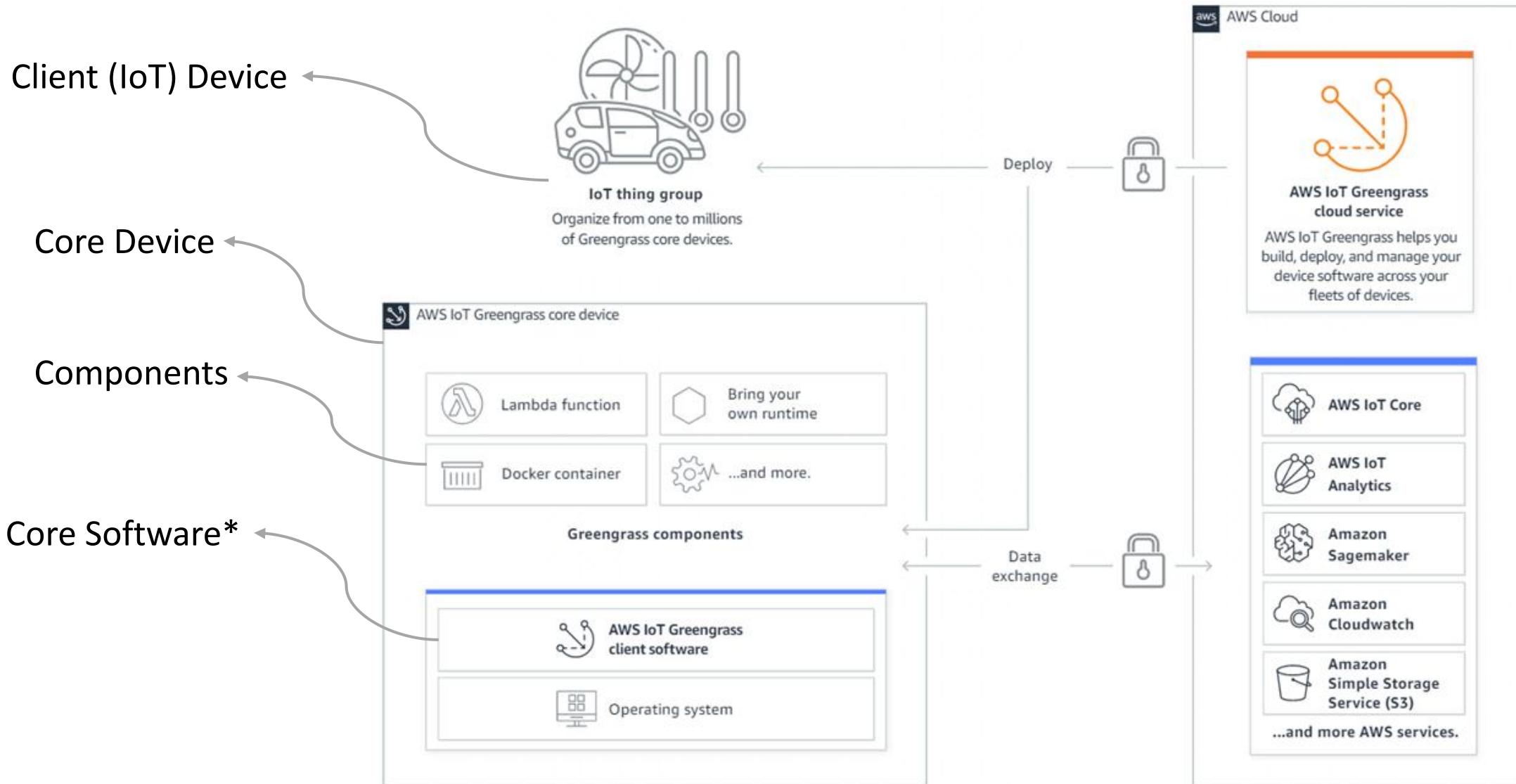


Software Concept



Action/Process

# How AWS IoT Greengrass works



\* Greengrass *core software* is also called Greengrass *client software* in AWS documentation. Client devices use the IoT Device SDK

# 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	Linux	Windows	
Deploy and invoke Lambda functions	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	
Publish messages to Amazon Simple Notification Service using the Amazon SNS component	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	
Amazon Kinesis Data Firehose delivery streams using the Kinesis Data Firehose component	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	
Amazon Kinesis Video Streams using the edge connector for Kinesis Video Streams component	<input checked="" type="checkbox"/> Yes	-	
Installation		Linux	Windows
Feature			
Run AWS IoT Greengrass in a Docker container using a prebuilt Docker image	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	

Remote maintenance and updates		Linux	Windows
Feature			
Manage core devices with AWS Systems Manager			
Connect to core devices with AWS Systems Manager			
Machine learning		Linux	Windows
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:
  - Deploy and invoke components and Lambda functions.
  - Support MQTT messaging between AWS IoT and components.
  - Interact with local devices that connect and communicate over MQTT.
  - Execution of ML models on edge devices, enabling low-latency, real-time analytics.
  - Efficient and reliable high-volume IoT data transfer to the AWS Cloud using Stream Manager.
  - Perform secure, over-the-air (OTA) software updates.
  - Provide secure, encrypted storage of local secrets.
  - And more...

# Steps for Greengrass Core SW Installation

1

Set up your environment and install the required tools

2

Create temporary AWS credentials (using a temporary user)

3

Set up a Greengrass Core device in the AWS Management Console

4

Install the Greengrass Core sw using the temporary user's credentials

5

Remove the temporary user after installation

# Step 1: Set Up Your Environment

## For Linux Based Devices

- AWS IoT Greengrass Core software requires Java version 8 or higher.
- 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
```

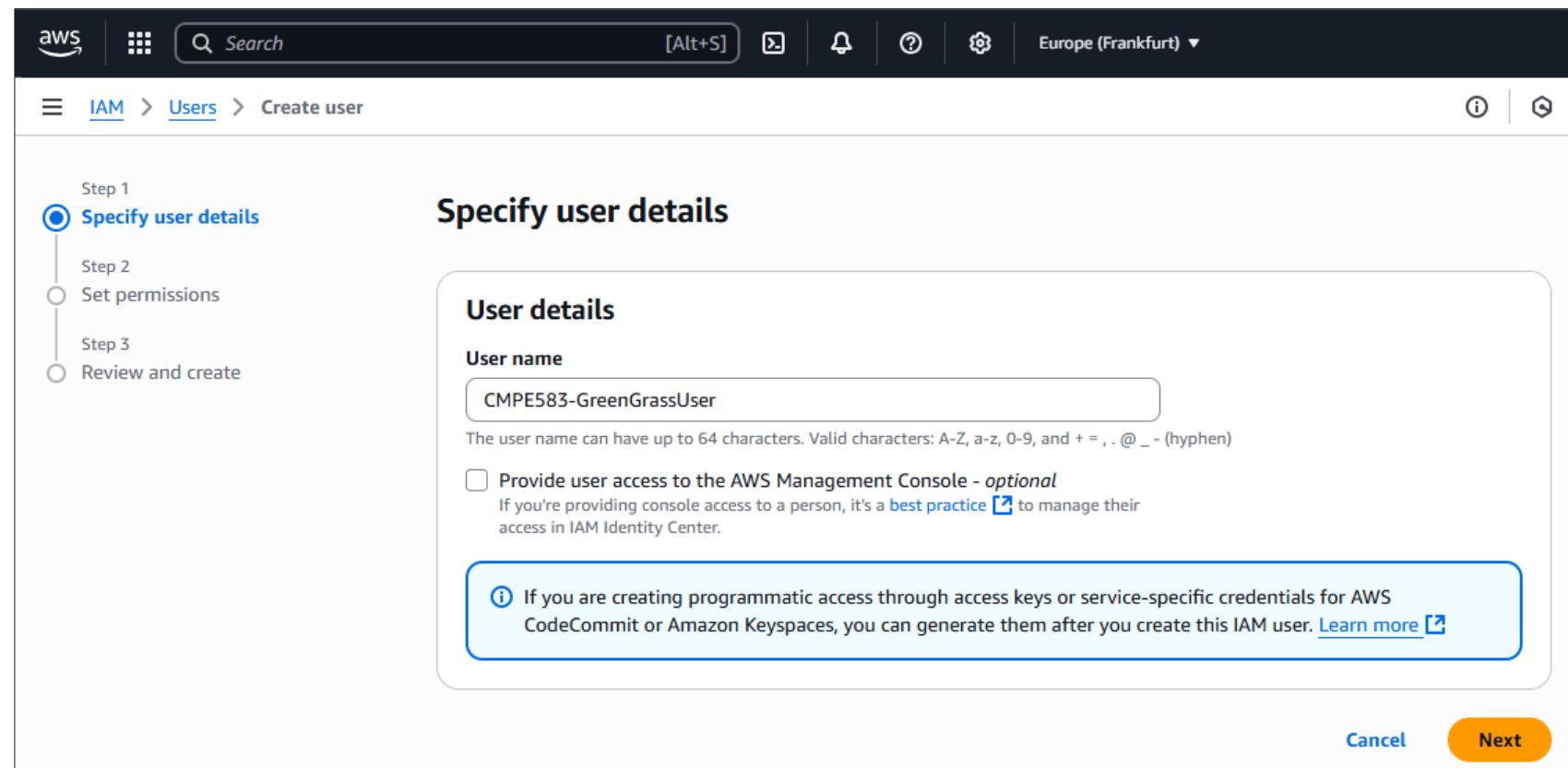
For Amazon Linux 2023 (AL2023):

```
sudo dnf install java-11-amazon-corretto -y
```

# 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.



# Step 2: Create AWS Credentials

## Create Access Keys for the User

The screenshot shows the AWS IAM User Details page for a user named 'CMPE583-GreenGrassUser'. The left sidebar shows the 'Access management' section with 'Users' selected. The main content area displays the 'Access keys (0)' section, which includes a note about avoiding long-term credentials and a 'Create access key' button. An orange arrow points to this button.

Identity and Access Management (IAM)

No MFA devices. Assign an MFA device to improve the security of your AWS environment

Assign MFA device

Access keys (0)

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

## Define Access Key Use-Case

The screenshot shows the AWS IAM 'Create access key' interface. On the left, a sidebar lists steps: Step 1 (selected, titled 'Access key best practices & alternatives'), Step 2 - optional ('Set description tag'), and Step 3 ('Retrieve access keys'). The main content area is titled 'Access key best practices & alternatives' and includes a note about avoiding long-term credentials. It then lists three use cases:

- Use case**
  - Command Line Interface (CLI)**  
You plan to use this access key to enable the AWS CLI to access your AWS account.
  - Local code**  
You plan to use this access key to enable application code in a local development environment to access your AWS account.
  - Application running on an AWS compute service**  
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.

An orange arrow points to the 'Command Line Interface (CLI)' option.

# Step 2: Create AWS Credentials

## Copy the Access Keys

The screenshot shows the AWS IAM 'Create access key' page for a user named 'CMPE583-GreenGrassUser'. The navigation path is: IAM > Users > CMPE583-GreenGrassUser > Create access key. On the left, a sidebar lists steps: Step 1 (Access key best practices & alternatives), Step 2 - optional (Set description tag), Step 3 (Retrieve access keys), and the current step, Step 4 (Retrieve access keys, highlighted in blue). The main content area is titled 'Retrieve access keys' and contains an 'Access key' section. It states: 'If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.' Below this, there are two fields: 'Access key' containing the value 'AKIAWY5JNGIBEX432LCK' and 'Secret access key' containing a redacted value '\*\*\*\*\*' followed by a 'Show' link. At the bottom, there is a section titled 'Access key best practices' with the following 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.

# 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 'Manage' section with various options like 'All devices', 'Greengrass devices' (which is expanded to show 'Core devices', 'Components', 'Deployments', 'Groups (V1)', 'LPWAN devices', 'Software packages', 'Remote actions', 'Message routing', 'Retained messages', 'Security', and 'Fleet Hub'). The main area is titled 'Greengrass core devices' with an 'Info' link. It displays 'Greengrass core devices (0)'. Below this, there's a note: 'An AWS IoT Greengrass core device is a device that runs the Greengrass Core software. Once your device is provisioned, it will show up here.' A search bar says 'Find core devices' with a page number '1'. At the bottom, there's a table header with columns 'Name', 'Status', 'Runtime', and 'Status reported'. A message says 'No core devices' and 'You don't have any Greengrass core devices in eu-central-1.'. A blue 'Set up one core device' button is at the bottom. An orange arrow points to the 'Set up core device' button in the top right.

# Step 3: Set Up a Greengrass Core Device in AWS

The screenshot shows the AWS IoT Greengrass Core devices setup process. It consists of two main sections: Step 1 and Step 2.

**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 3: Set Up a Greengrass Core Device in AWS

## Select Your Runtime

The screenshot shows the AWS IoT Greengrass Core devices setup process. The navigation bar at the top includes links for AWS IoT, Greengrass, Core devices, and Set up one Greengrass core device. Below the navigation is a title 'Step 3: Install the Greengrass Core software'. A sub-section titled 'Greengrass Core software runtime' has a 'Info' link. A note says 'Select the Core software runtime to be installed onto your device.' Two options are shown: 'Nucleus Classic' (selected) and 'Nucleus Lite'. The 'Nucleus Classic' section describes it as a powerful runtime compatible with all Greengrass components, working with Linux and Windows. The 'Nucleus Lite' section describes it as designed for lower memory and storage requirements, with limited compatibility and working only with Linux. Below these sections is an 'Operating system' section with 'Linux' and 'Windows' radio buttons. At the bottom are 'Cancel' and 'View core devices' buttons.

AWS IoT > Greengrass > Core devices > Set up one Greengrass core device

**Step 3: Install the Greengrass Core software**

**Greengrass Core software runtime** | [Info](#)

Select the Core software runtime to be installed onto your device.

**Nucleus Classic**  
Nucleus Classic is a powerful runtime that requires higher device memory and storage. It's compatible with all Greengrass components. Works with Linux and Windows devices.

**Nucleus Lite**  
Nucleus Lite is designed for devices with lower memory and storage requirements, enabling a wider range of use cases. It has limited compatibility with Greengrass components and only works with Linux devices.

**Operating system**

Linux

Windows

[Cancel](#) [View core devices](#)

# 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-nucleus-latest.zip && unzip greengrass-nucleus-latest.zip -d GreengrassInstaller
```

Run the installer:

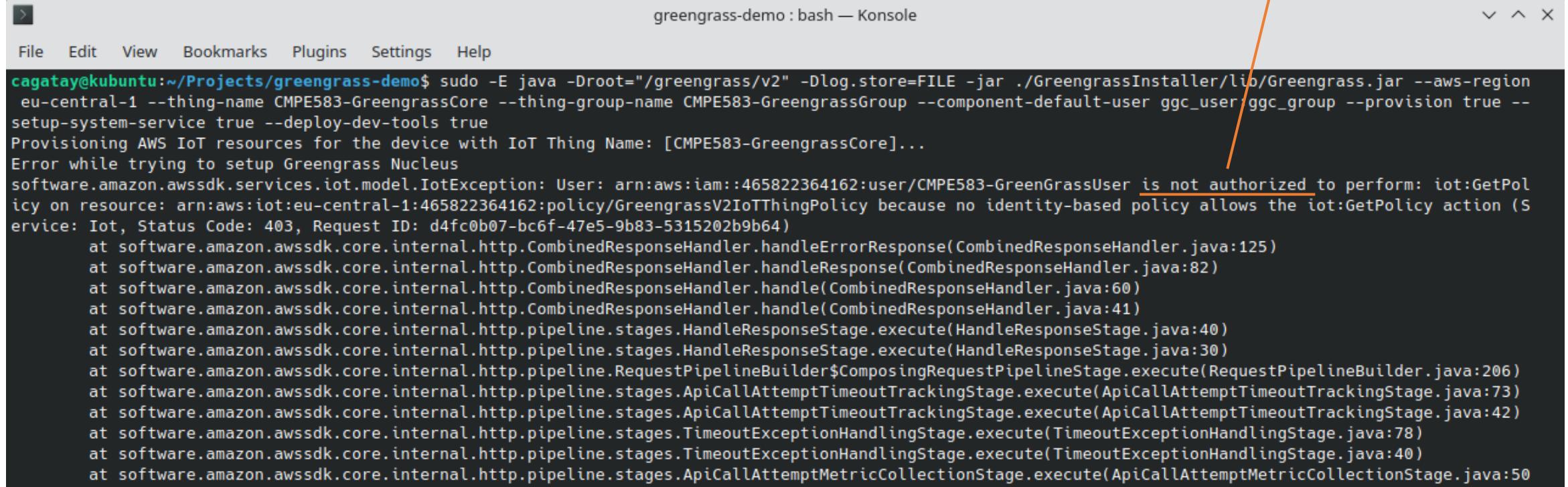
```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE -jar ./GreengrassInstaller/lib/Greengrass.jar --aws-region eu-central-1 --thing-name CMPE583-GreenGrassCore --thing-group-name CMPE583-GreenGrassGroup --component-default-user ggc_user:ggc_group --provision true --setup-system-service true --deploy-dev-tools 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 a standard Linux-style interface with a menu bar (File, Edit, View, Bookmarks, Plugins, Settings, Help) and a title bar. The terminal content 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

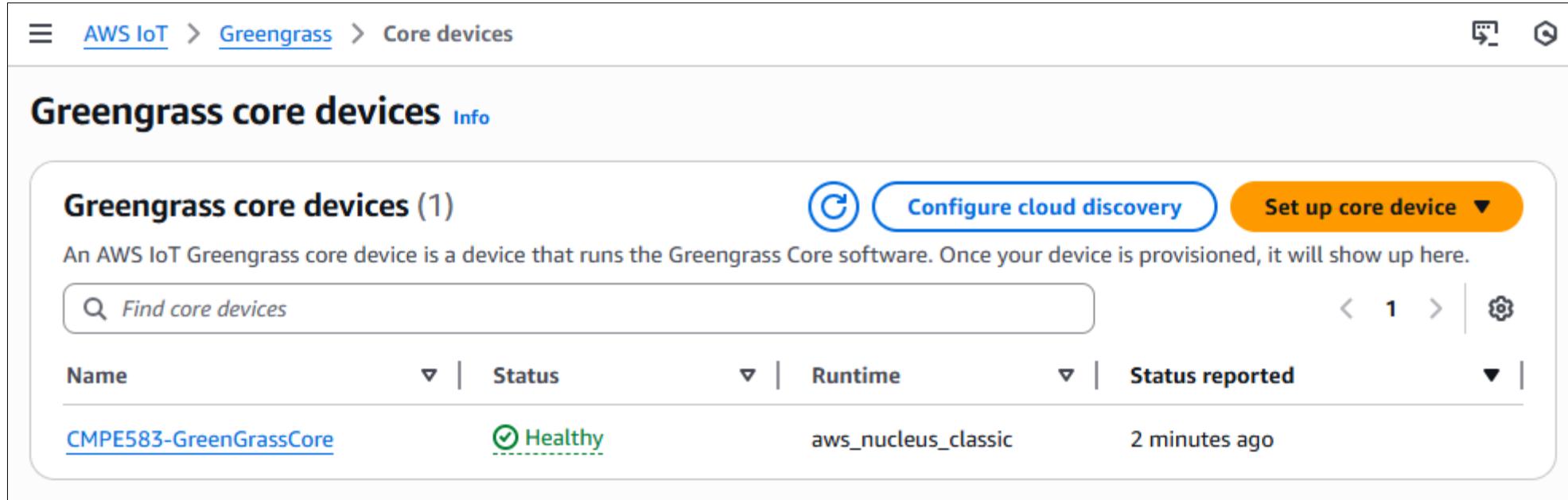
The screenshot shows the AWS IAM 'Create policy' wizard. The current step is 'Specify permissions'. The policy editor displays a JSON template for a provisioning policy:

```
1 ▼ {
2     "Version": "2012-10-17",
3     "Statement": [
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            ]
23        }
24    ]
25}
```

The 'Actions' tab is selected in the policy editor. A sidebar on the right allows adding new statements or editing existing ones.

# 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 page. The navigation bar at the top includes 'AWS IoT' and 'Greengrass'. The main title 'Greengrass core devices' has an 'Info' link. Below it, a summary box says 'Greengrass core devices (1)'. A note states: 'An AWS IoT Greengrass core device is a device that runs the Greengrass Core software. Once your device is provisioned, it will show up here.' There are buttons for 'Configure cloud discovery' and 'Set up core device'. A search bar labeled 'Find core devices' is present. The table below lists one device:

Name	Status	Runtime	Status reported
<a href="#">CMPE583-GreenGrassCore</a>	<span>Healthy</span>	aws_nucleus_classic	2 minutes ago

# Step 4: Install the Greengrass Core Software

## Check If GreengrassV2TokenExchangeRole Is Created

The screenshot shows the AWS IAM Roles page with the role 'GreengrassV2TokenExchangeRole' selected. The role is described as 'Role for Greengrass IoT things to interact with AWS services using token exchange service'. The 'Permissions' tab is selected, showing one policy named 'GreengrassV2TokenExchangeRoleAccess'. The policy document is displayed below:

```
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

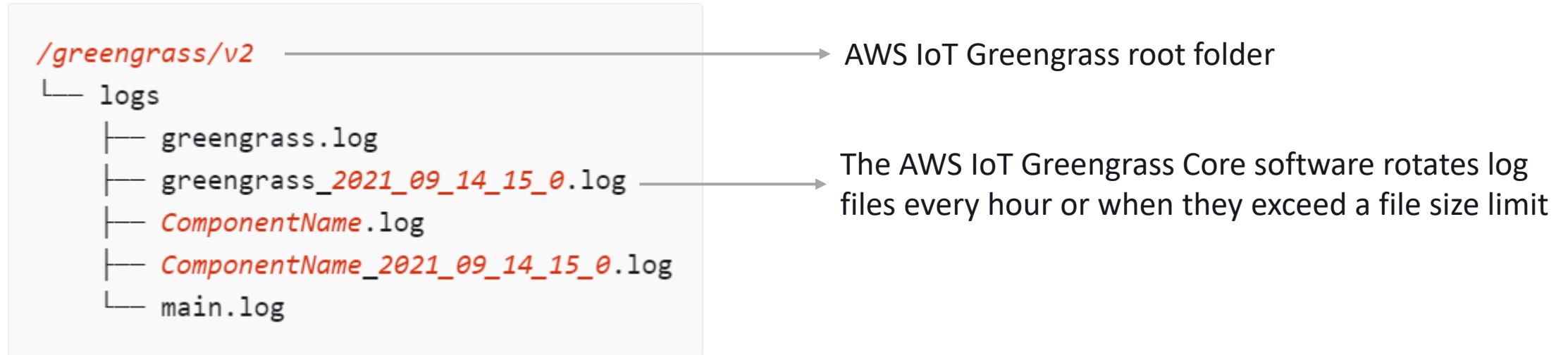
- If the GreengrassInstaller cannot create the GreengrassV2TokenExchangeRole and prints an error message, you can manually create a custom role in IAM console with the following JSON value and reinstall the core software:

```
{  
    "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

# Steps for Greengrass Component Deployment

1

Upload artifacts  
to Amazon S3

2

Set S3 access  
policies

3

Create the  
component and  
configure its  
recipe

4

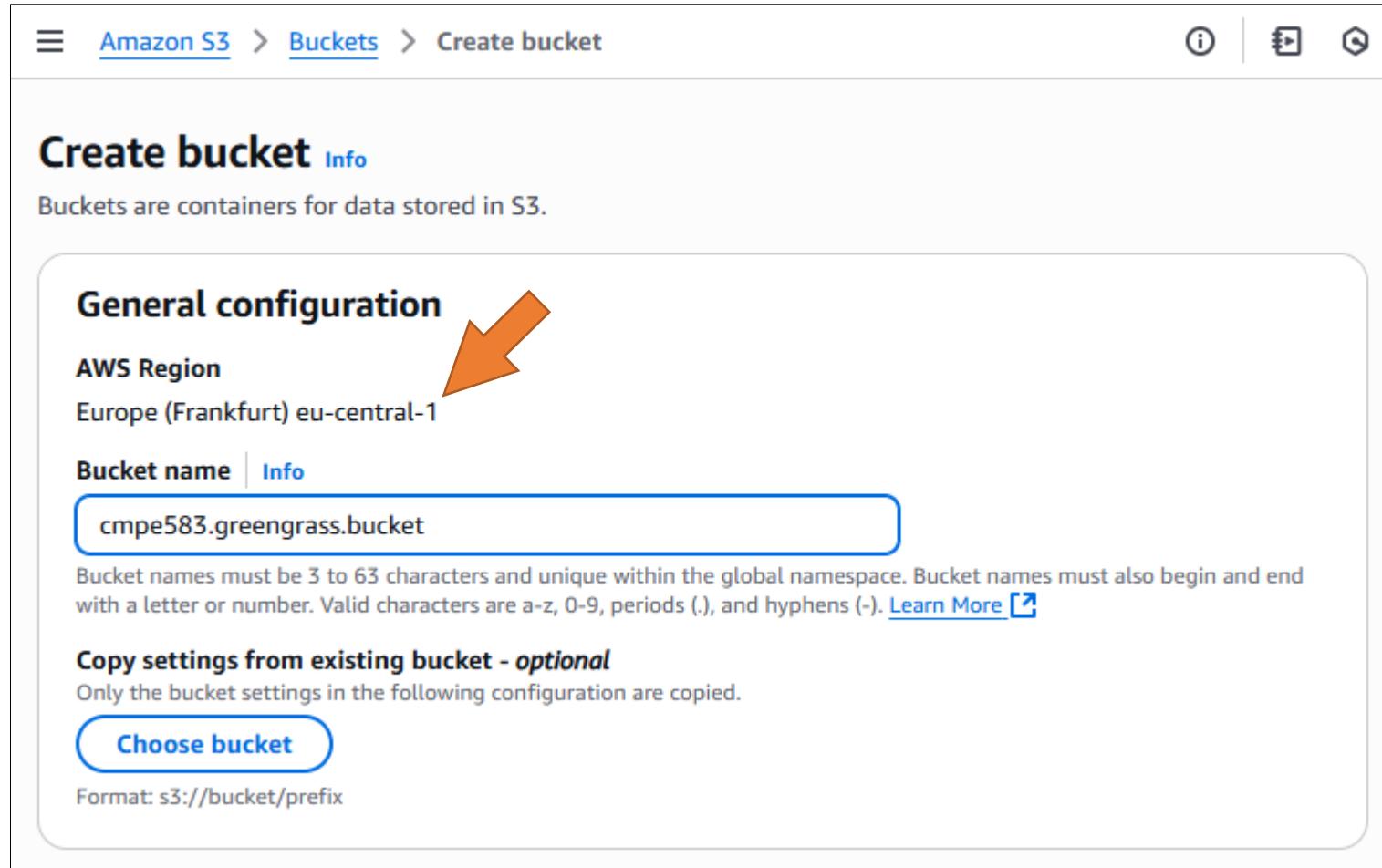
Deploy the  
component via  
the AWS  
Management  
Console

5

Verify  
deployment in  
the console and  
on the device  
terminal

# Step 1: Upload the Artifacts

## Create a bucket via AWS S3 (Console)



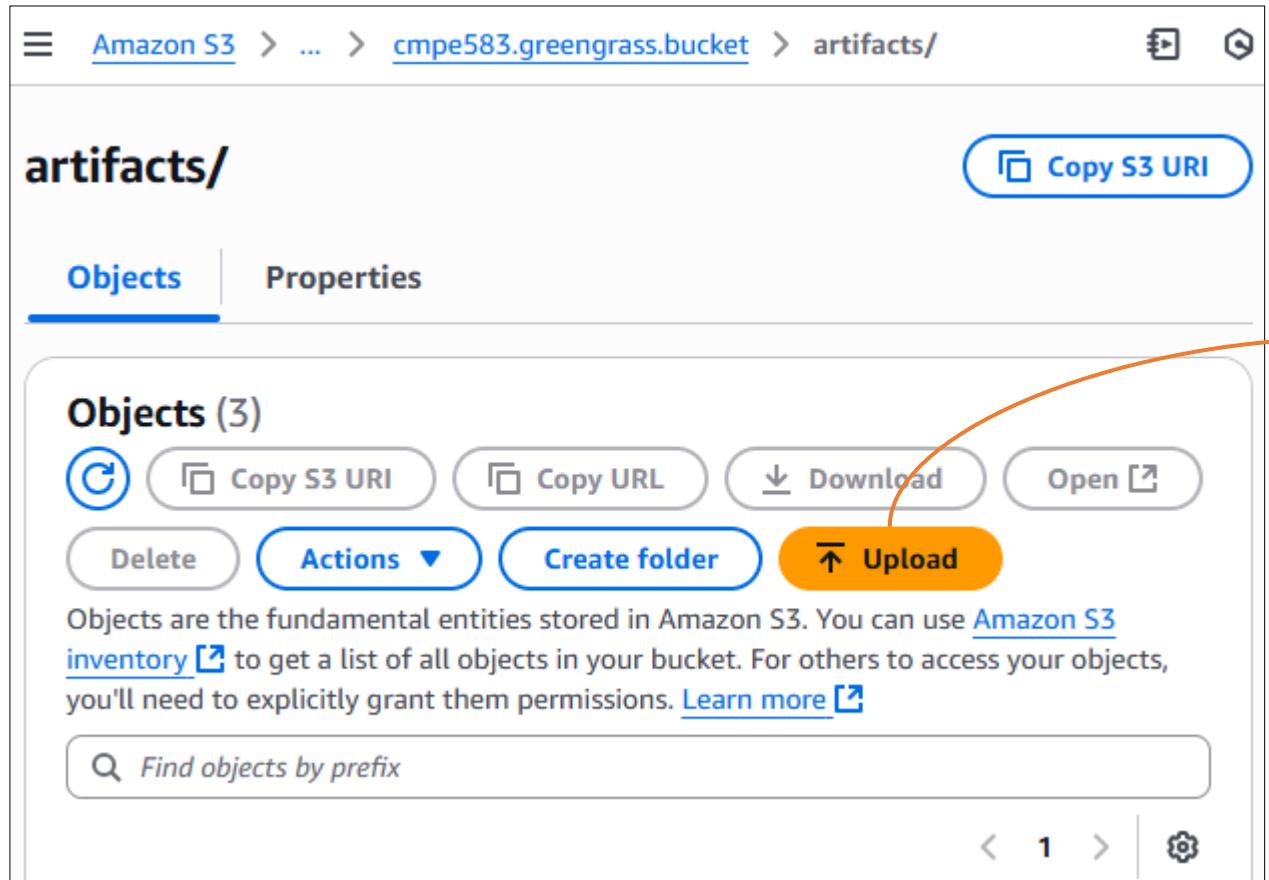
The screenshot shows the 'Create bucket' page in the AWS S3 console. The URL in the top navigation bar is [Amazon S3](#) > [Buckets](#) > [Create bucket](#). The main title is 'Create bucket' with an 'Info' link. Below it, a sub-instruction says 'Buckets are containers for data stored in S3.' A large orange arrow points to the 'AWS Region' dropdown menu, which is set to 'Europe (Frankfurt) eu-central-1'. The 'Bucket name' field contains 'cmpe583.greengrass.bucket'. A note below the field states: 'Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn More](#)'.

- 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 1: Upload the Artifacts

Create artifacts folder & upload files

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



```
cmpe583-PrintMsg.py x
S3 > artifacts > 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",  
                "s3:PutObject",  
                "s3:PutObjectAcl"  
            ],  
            "Resource": "arn:aws:s3:::cmpe583.greengrass.bucket/*"  
        }  
    ]  
}
```

# 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.

The screenshot shows two panels. The left panel is the 'GreengrassV2TokenExchangeRole' details page under the 'Permissions' tab. It displays the ARN (arn:aws:iam::465822364162:role/GreengrassV2TokenExchangeRole) and a maximum session duration of 1 hour. The 'Add permissions' button is highlighted with a red box and an orange arrow points from it to the right panel. The right panel is the 'Specify permissions' JSON editor. It shows the following JSON code:

```
1  "Version": "2012-10-17",
2  "Statement": [
3    {
4      "Effect": "Allow",
5      "Action": [
6        "s3:GetObject",
7        "s3:PutObject",
8        "s3:PutObjectAcl"
9      ],
10     "Resource": "arn:aws:s3:::cmpe583.greengrass.bucket/*"
11   }
12 ]
13 }
14 }
```

# Step 3: Create the Component

## Via AWS IoT Greengrass (Console)

The screenshot shows the AWS IoT Greengrass Components page. The left sidebar has a 'Components' section selected. The main area displays 'Greengrass components' with tabs for 'My components' (selected), 'Public components', and 'Community components'. An orange arrow points to the 'Create component' button in the 'My components' section. The page indicates there are 0 components and provides a search bar and pagination controls.

AWS IoT > Greengrass > Components

Manage

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

## Greengrass components Info

**My components** Public components Community components

### My components (0)

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

Find components

Name	Version	Supported runtime	Operating system
No components			

You don't have any Greengrass components in eu-central-1.

[Create component](#)

# Step 3: Create the Component

## Define the Recipe

The screenshot shows the 'Create component' page in the AWS IoT Greengrass console. The navigation bar at the top includes 'AWS IoT > Greengrass > Components > Create component'. The main title 'Create component' is followed by a sub-instruction: '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.' Below this, the 'Component information' section is displayed. It contains three options for defining the component source:

- 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": "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://cmpe583.greengrass.bucket/artifacts/print_msg.py" } ]  
    }  
  ]  
}
```

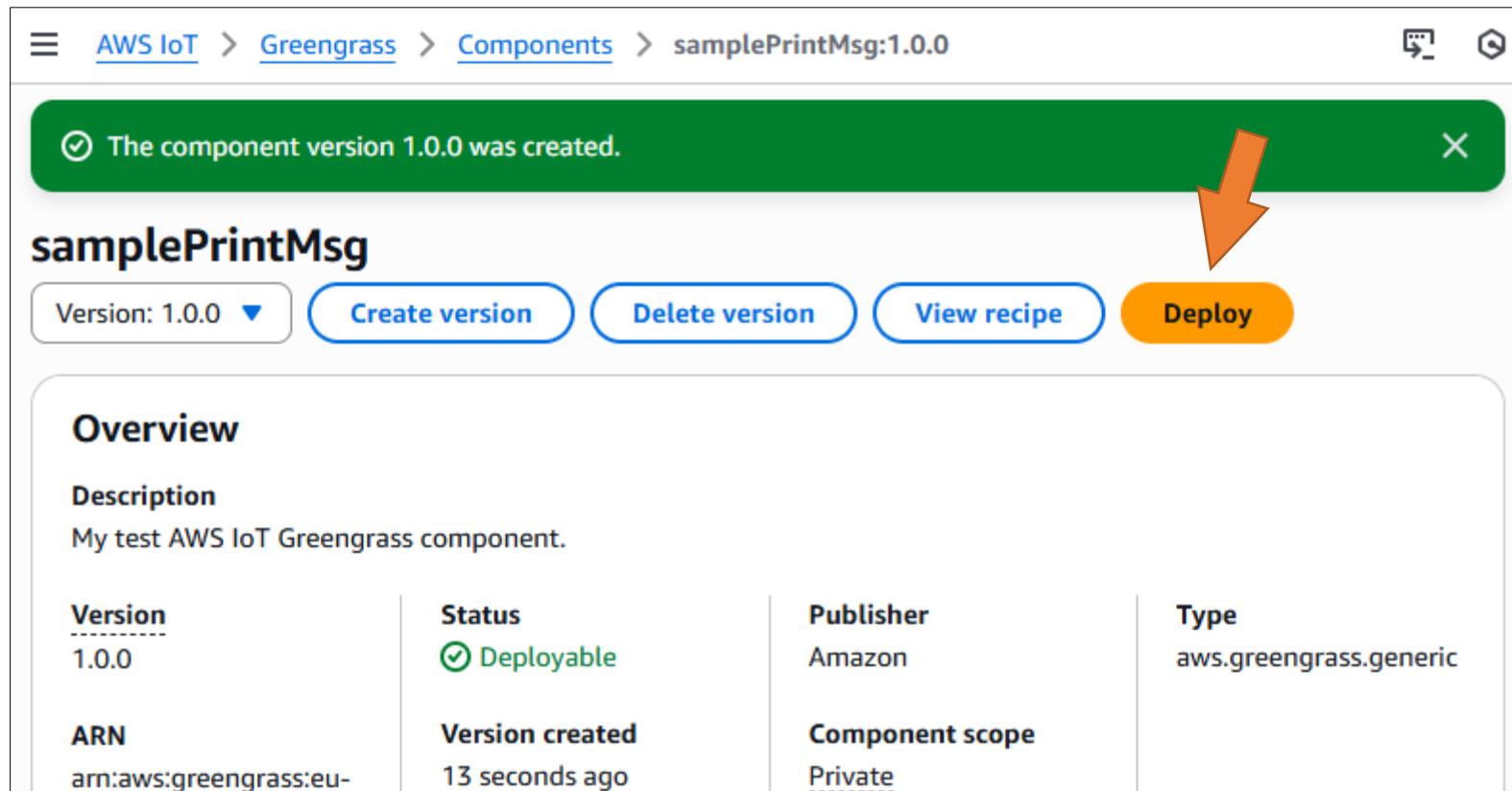
This test will be run on Linux device. So other OSs are ignored!

Define how to execute your artifact

Provide the artifact URI

# Step 4: Deploy the Component

## Via AWS IoT Greengrass (console)



# Step 4: Deploy the Component

## Via AWS IoT Greengrass (console)

The screenshot shows the AWS IoT Greengrass console interface. At the top, there's a navigation bar with tabs like 'samplePrintMsg', 'Version: 1.0.0', 'Create version', 'Delete version', 'View recipe', and a prominent 'Deploy' button. Below this, a modal window titled 'Add to deployment' is open. Inside, under the 'Deployment' section, the 'Add to existing deployment' radio button is selected, while 'Create new deployment' is unselected. A search bar labeled 'Find by deployment name or target name' is present. The main list displays a single item: 'Deployment for CMPE583-GreenGrassGroup'. This item includes details: Target name 'CMPE583-GreenGrassGroup', Target type 'Thing group', Status 'Active', and last updated '31 minutes ago'. At the bottom of the modal, there are 'Cancel' and 'Next' buttons, with an orange arrow pointing to the 'Next' button.

Deployment	Target name	Target type	Status	Deployment created
Deployment for CMPE583-GreenGrassGroup	CMPE583-GreenGrassGroup	Thing group	Active	31 minutes ago

Cancel Next

# Step 4: Deploy the Component

## Configure Deployment

The screenshot shows the AWS IoT Greengrass 'Specify target' deployment step. The navigation bar at the top includes 'AWS IoT > Greengrass > Deployments > f0bcd905-6333-4c55-858c-31521f022bdb > Revise deployment'. On the left, a vertical navigation menu lists steps: Step 1 (Specify target, highlighted with a blue circle), Step 2 - optional (Select components), Step 3 - optional (Configure components), Step 4 - optional (Configure advanced settings), and Step 5 (Review). The main content area is titled 'Specify target' and contains two sections: 'Deployment information' (with a 'Name - optional' field containing 'Test Deployment for CMPE583-GreenGrassGroup') and 'Deployment target' (with a 'Target name' dropdown containing 'CMPE583-GreenGrassGroup').

Step 1  
Specify target

Step 2 - optional  
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-GreenGrassGroup

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 name**

CMPE583-GreenGrassGroup ▾

# Step 4: Deploy the Component

## Select the Component

AWS IoT > Greengrass > Deployments > f0bcd905-6333-4c55-858c-31521f022bdb > 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) Info**

Name	Supported runtime	Operating system	Architecture
samplePrintMsg	aws_nucleus_classic	linux	All

**Public components (48) Info**

Name	Supported runtime	Operating system	Architecture
aws.greengrass.Cli	aws_nucleus_classic	linux, darwin, windows	All

Cancel Skip to Review Previous Next



# Step 4: Deploy the Component

## Configure Advanced Settings

The screenshot shows the AWS IoT Greengrass Deployment configuration interface. The navigation bar at the top includes links for AWS IoT, Greengrass, Deployments, and a specific deployment ID (f0bcd905-6333-4c55-858c-31521f022bdb). The sub-page title is "Revise deployment". On the left, a vertical navigation menu lists steps: Step 1 (Specify target), Step 2 - optional (Select components), Step 3 - optional (Configure components), Step 4 - optional (Configure advanced settings, highlighted with a blue circle), and Step 5 (Review). The main content area is titled "Configure advanced settings - optional" and contains four sections: "Rollout configuration", "Timeout configuration", "Cancel configuration", and "Deployment policies". Each section has a brief description. A large orange arrow points downwards from the "Deployment policies" section towards the "Next" button at the bottom right. The bottom navigation bar includes "Cancel", "Previous", and "Next" buttons.

AWS IoT > Greengrass > Deployments > f0bcd905-6333-4c55-858c-31521f022bdb > 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

**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 device

Cancel Previous Next

# Step 4: Deploy the Component

## Review & Deploy

AWS IoT > Greengrass > Deployments > f0bcd905-6333-4c55-858c-31521f022bdb > 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

### Review

#### Step 1: Deployment info

**Edit**

#### Deployment target

**Name**  
Test Deployment for CMPE583-GreenGrassGroup

**Target type**  
Thing group

**Target name**  
CMPE583-GreenGrassGroup

Download as JSON

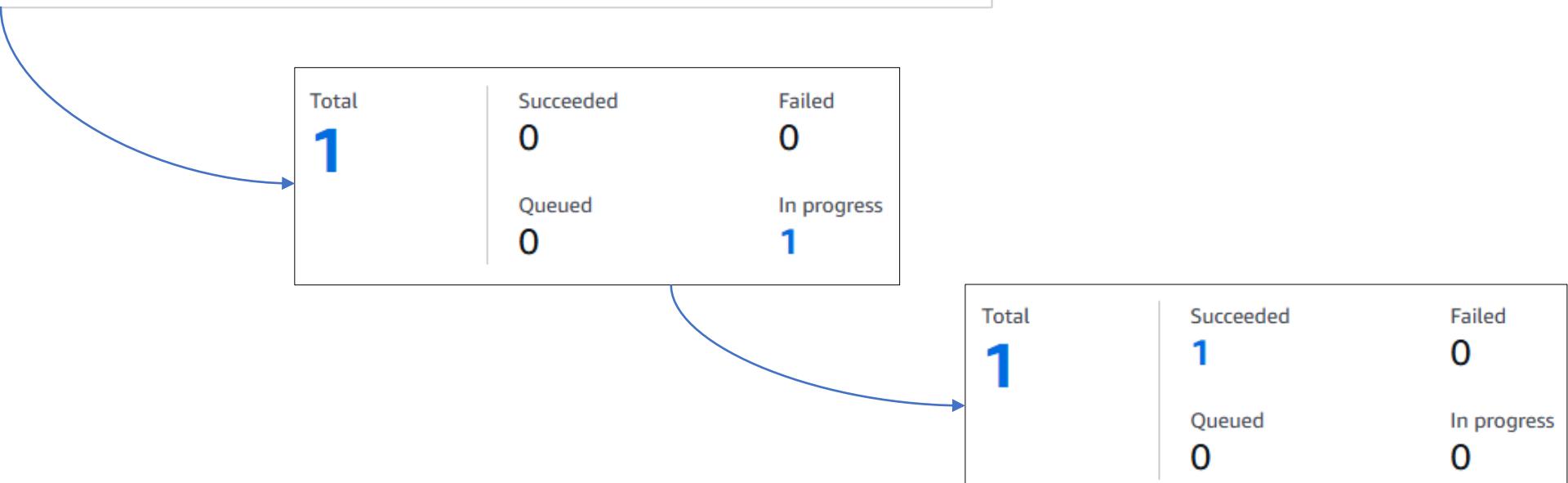
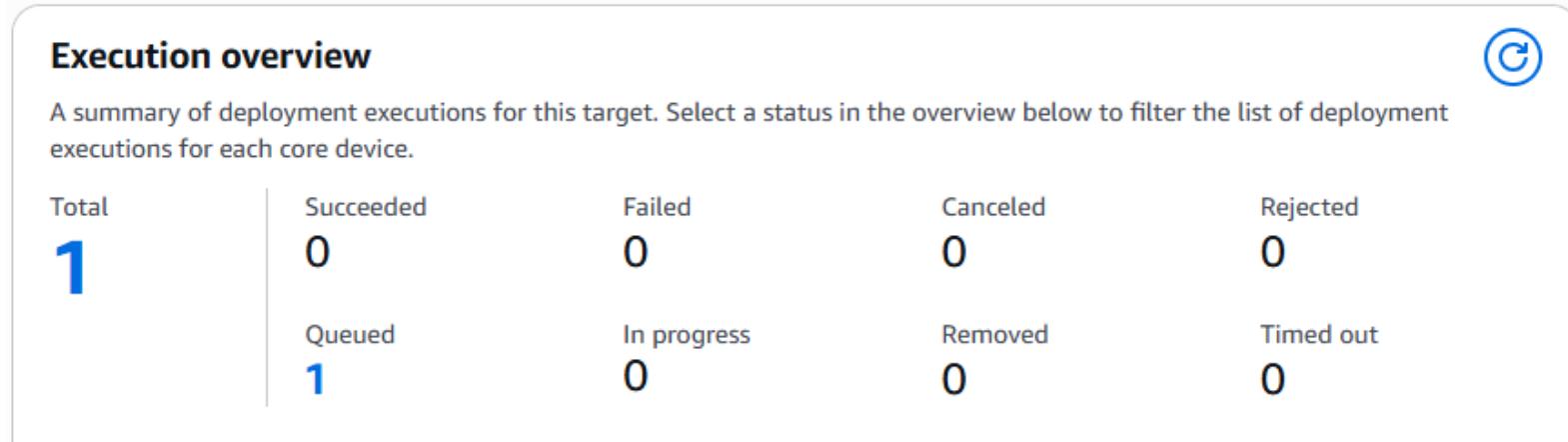
Cancel

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-Prin..."]}
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

## Overview

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

<b>Target</b> <a href="#">CMPE583-GreenGrassGroup</a>	<b>Target type</b> Thing group	<b>Deployment created</b> 8 minutes ago
<b>IoT job</b> <a href="#">bf4dbe22-ff26-41eb-acf2-956c4332ffc0</a>	<b>Deployment status</b> <span>Active</span>	

**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 <b>1</b>	Succeeded 0	Failed <b>1</b> 	Canceled 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 IAM 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 (48)

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](#)

Find components

1 match

ipdetector X | Clear filters

< 1 > | C

Name	Version	Supported runtime	Operating system	Architecture	Publisher
<a href="#">aws.greengrass.clientdevices.IPDetector</a>	2.2.2	aws_nucleus_classic	All	All	AWS

# Collecting IP Addresses of Core Devices

## Deploy an IPDetector Component

The screenshot shows the AWS IoT Greengrass Components page for the `aws.greengrass.clientdevices.IPDetector` component. The component is version 2.2.2, ARN is `arn:aws:greengrass:eu-central-1:aws:components:aws.greengrass.clientdevices.IPDetector:versions:2.2.2`, Status is Deployable, Version created was 4 months ago, Publisher is AWS, Component scope is Public, and Type is `aws.greengrass.plugin`. The Dependencies section lists `aws.greengrass.Nucleus` as a dependency with a version requirement of `>=2.2.0 <2.16.0` and a dependency type of Soft.

Component	Version requirement	Dependency type
<code>aws.greengrass.Nucleus</code>	<code>&gt;=2.2.0 &lt;2.16.0</code>	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!

**aws.greengrass.clientdevices.IPDetector**

Version: 2.2.2 ▾ View recipe Deploy

**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
<a href="#">Test Deployment for CMPE583-GreenGrassGroup</a>	CMPE583-GreenGrassGroup	Thing group	Active	6 hours ago

Cancel Next

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

Component	Version requirement	Dependency type
-----------	---------------------	-----------------

# Collecting IP Addresses of Core Devices

## Select All Components You Want to Deploy

The screenshot shows the 'Select components - optional' step in the AWS IoT Greengrass 'Revise deployment' process. The left sidebar lists steps: 'Specify target', 'Step 2 - optional', 'Select components' (which is active), 'Step 3 - optional', 'Configure components', 'Step 4 - optional', 'Configure advanced settings', 'Step 5', and '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	Supported runtime	Operating system
<a href="#">samplePrintMsg</a>	aws_nucleus_classic	linux

**Public components (48)**

Name	Supported runtime	Operating system
<a href="#">aws.greengrass.clientdevices.IDPDetector</a>	aws_nucleus_classic	All
<a href="#">aws.greengrass.Cli</a>	aws_nucleus_classic	linux, darwin, windows

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.aws.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 IAM Roles management interface. On the left, a sidebar menu under 'Identity and Access Management (IAM)' includes 'Dashboard', 'Access management' (with 'Roles' selected), 'Policies', 'Identity providers', and 'Account settings'. The main panel title is 'Roles (55) Info', with a 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.' It features a search bar and navigation links. A large orange arrow points to the 'Create role' button at the top right of the main content area.

<input type="checkbox"/>   Role name	▲   Trusted entities
<a href="#">AWSServiceRoleForAmazonSSM</a>	AWS Service: ssm (Service-Linked Role)
<a href="#">AWSServiceRoleForAutoScaling</a>	AWS Service: autoscaling (Service-Linked Role)
<a href="#">AWSServiceRoleForCloudFormationStackSetsOrgMe...</a>	AWS Service: member.org.stacksets
<a href="#">AWSServiceRoleForCloudFrontLogger</a>	AWS Service: logger.cloudfront (Service-Linked Role)
<a href="#">AWSServiceRoleForCloudTrail</a>	AWS Service: cloudtrail (Service-Linked Role)

# Troubleshooting

## Create an IAM Role

The screenshot shows the AWS IAM 'Create role' wizard. The navigation bar at the top says 'IAM > Roles > Create role'. On the left, a sidebar shows 'Step 1 Select trusted entity' (highlighted with a blue circle), 'Step 2 Add permissions' (gray circle), and 'Step 3 Name, review, and create' (gray circle). The main content area is titled 'Select trusted entity' with an 'Info' link. It has a sub-section 'Trusted entity type' with five options:

- 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.
- Custom trust policy**: Create a custom trust policy to enable others to perform actions in this account.

Below this is a 'Use case' section with the sub-section 'Service or use case'. A dropdown menu shows 'Greengrass' (selected) and a downward arrow icon. The text 'Allow an AWS service like EC2, Lambda, or others to perform actions in this account.' is displayed. At the bottom, there's a 'Choose a use case for the specified service.' section with a 'Use case' heading and a single option selected:

- 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.

The screenshot shows the AWS IAM 'Create role' wizard at Step 2: 'Add permissions'. The left sidebar lists steps: Step 1 (Select trusted entity), Step 2 (Add permissions, which is selected and highlighted in blue), and Step 3 (Name, review, and create). The main area is titled 'Add permissions' and contains a search bar with 'greengrass', a filter dropdown set to 'All types', and a results table. The table has columns for 'Policy name' and 'Type'. It lists four policies: 'AWSGreengrassFullAccess' (AWS managed), 'AWSGreengrassReadOnlyAccess' (AWS managed), 'AWSGreengrassResourceAccessRolePolicy' (AWS managed, checked with a blue checkbox), and 'AWSIoTDeviceTesterForGreengrassFullAccess' (AWS managed).

Policy name	Type
<input type="checkbox"/> AWSGreengrassFullAccess	AWS managed
<input type="checkbox"/> AWSGreengrassReadOnlyAccess	AWS managed
<input checked="" type="checkbox"/> AWSGreengrassResourceAccessRolePolicy	AWS managed
<input type="checkbox"/> AWSIoTDeviceTesterForGreengrassFullAccess	AWS managed

# Troubleshooting

## Review and Create Role

The screenshot shows the AWS IAM 'Create role' wizard at Step 3: Name, review, and create. The left sidebar shows a vertical navigation path: Step 1 (Select trusted entity), Step 2 (Add permissions), Step 3 (Name, review, and create), and the current step, which is highlighted with a blue circle.

**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 letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: \_+=,. @-/\\ [{}]!#\$%^\*()\_;":`

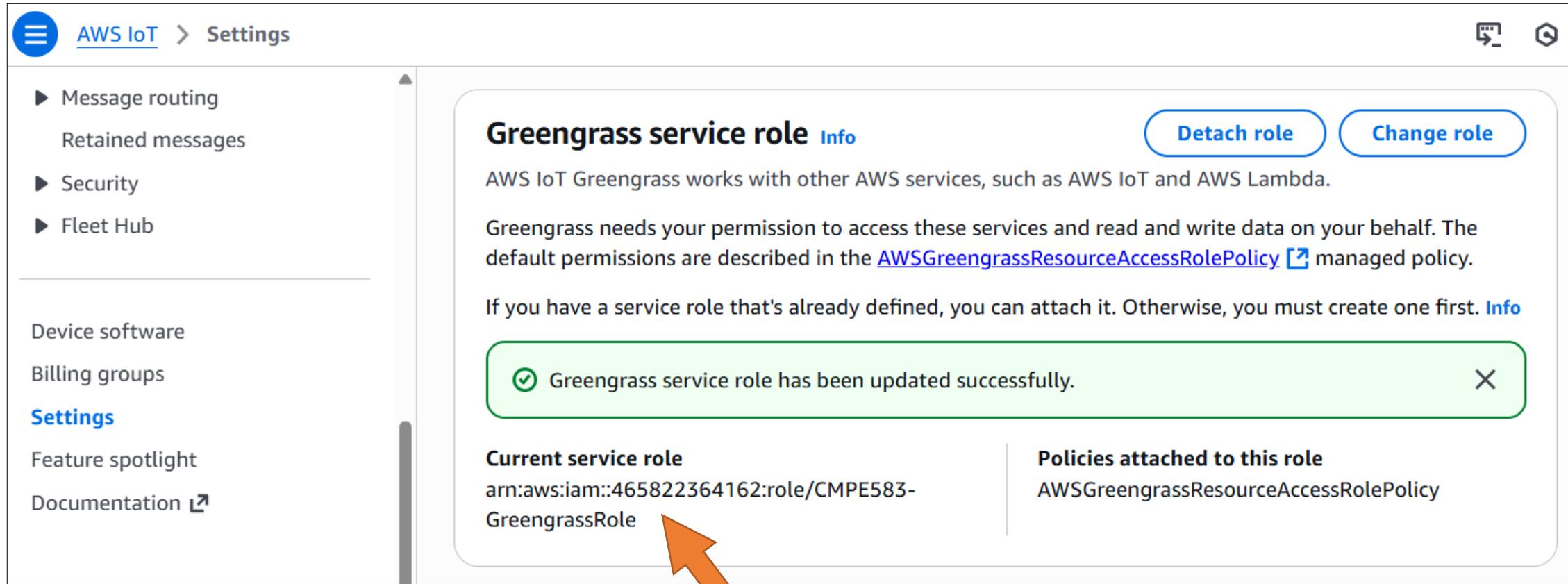
# Troubleshooting

## Attach Role to AWS IoT Service

The screenshot shows the AWS IoT Greengrass service role settings page. On the left, a sidebar lists navigation options: Message routing, Retained messages, Security, Fleet Hub, Device software, Billing groups, **Settings**, Feature spotlight, and Documentation. An orange arrow points to the 'Settings' link. The main content area is titled 'Greengrass service role' with an 'Info' link. It contains a description: 'AWS IoT Greengrass works with other AWS services, such as AWS IoT and AWS Lambda. 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.' Below this, there are sections for 'Current service role' (listing '-') and 'Policies attached to this role' (listing '-'). At the top right of the content area, there are 'Detach role' and 'Attach role' buttons; the 'Attach role' button is highlighted with an orange arrow.

# Troubleshooting

## Attach Role to AWS IoT Service



The screenshot shows the AWS IoT Greengrass service role configuration page. The left sidebar lists various settings like Message routing, Security, and Fleet Hub. The main content area is titled "Greengrass service role" and contains a success message: "Greengrass service role has been updated successfully." Below this, it shows the "Current service role" as "arn:aws:iam::465822364162:role/CMPE583-GreengrassRole". To the right, it lists "Policies attached to this role" as "AWSGreengrassResourceAccessRolePolicy". An orange arrow points to the "Current service role" text.

AWS IoT > Settings

Greengrass service role [Info](#)

Detach role Change role

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

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 screenshot shows two main sections. On the left, a summary box displays deployment statistics: Total 1 (Succeeded 1, Failed 0, Queued 0, In progress 0). An orange arrow points from this box to the 'Client devices' tab on the right. The right section is a detailed view of a core device named 'CMPE583-GreenGrassCore'. It includes tabs for Components, Deployments, Thing groups, Client devices (which is selected), and Tags. Under the Client devices tab, there's a 'Cloud discovery configuration' section with a 'Configure cloud discovery' button. Below it is an 'MQTT broker endpoints (1)' section with a 'Manage endpoints' button, listing one endpoint at 10.0.2.15:8883. The top navigation bar shows the path: AWS IoT > Greengrass > Core devices > CMPE583-GreenGrassCore.

Total	Succeeded	Failed
1	1	0
Queued	In progress	
0	0	

AWS IoT > Greengrass > Core devices > CMPE583-GreenGrassCore

Components Deployments Thing groups **Client devices** Tags

**Cloud discovery configuration** Info [Configure cloud discovery](#)

With cloud discovery, core devices store their connectivity information in the AWS IoT Greengrass cloud service. Client devices connect to AWS IoT Greengrass to discover connectivity information for core devices where they can connect. Client devices must be AWS IoT things to use cloud discovery.

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

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.

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.
- AWS IoT Greengrass provides an improved version of the IPC client: IPC client V2
- 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.

# Steps for Deploying a Component Using Core IPC

1

Develop client code using Core MQTT IPC and upload artifacts to S3

2

Create the component and define its recipe

3

Set access control for MQTT topics and deploy the component

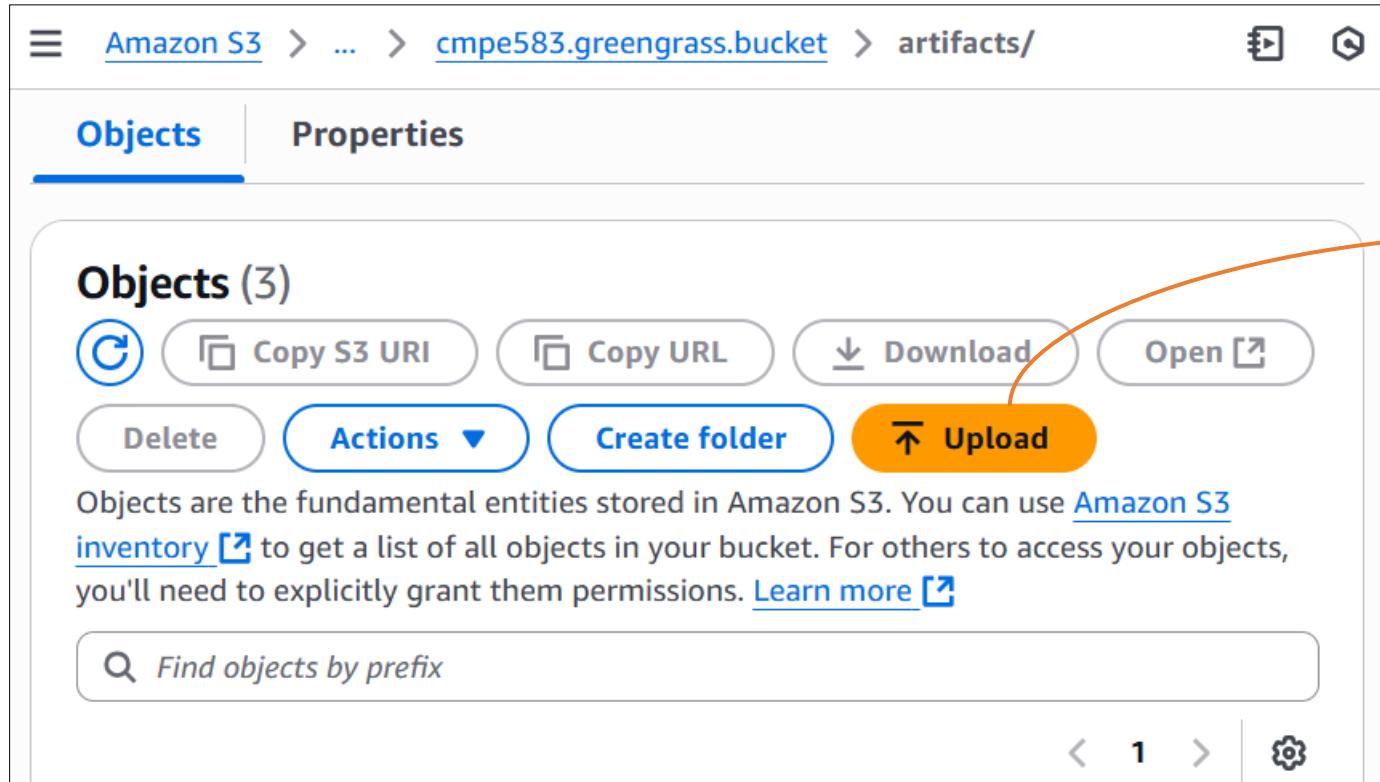
4

Verify and test the component using AWS MQTT Client

# Step 1: Develop & Upload Client-Side Code

## Via AWS IoT Greengrass (Console)

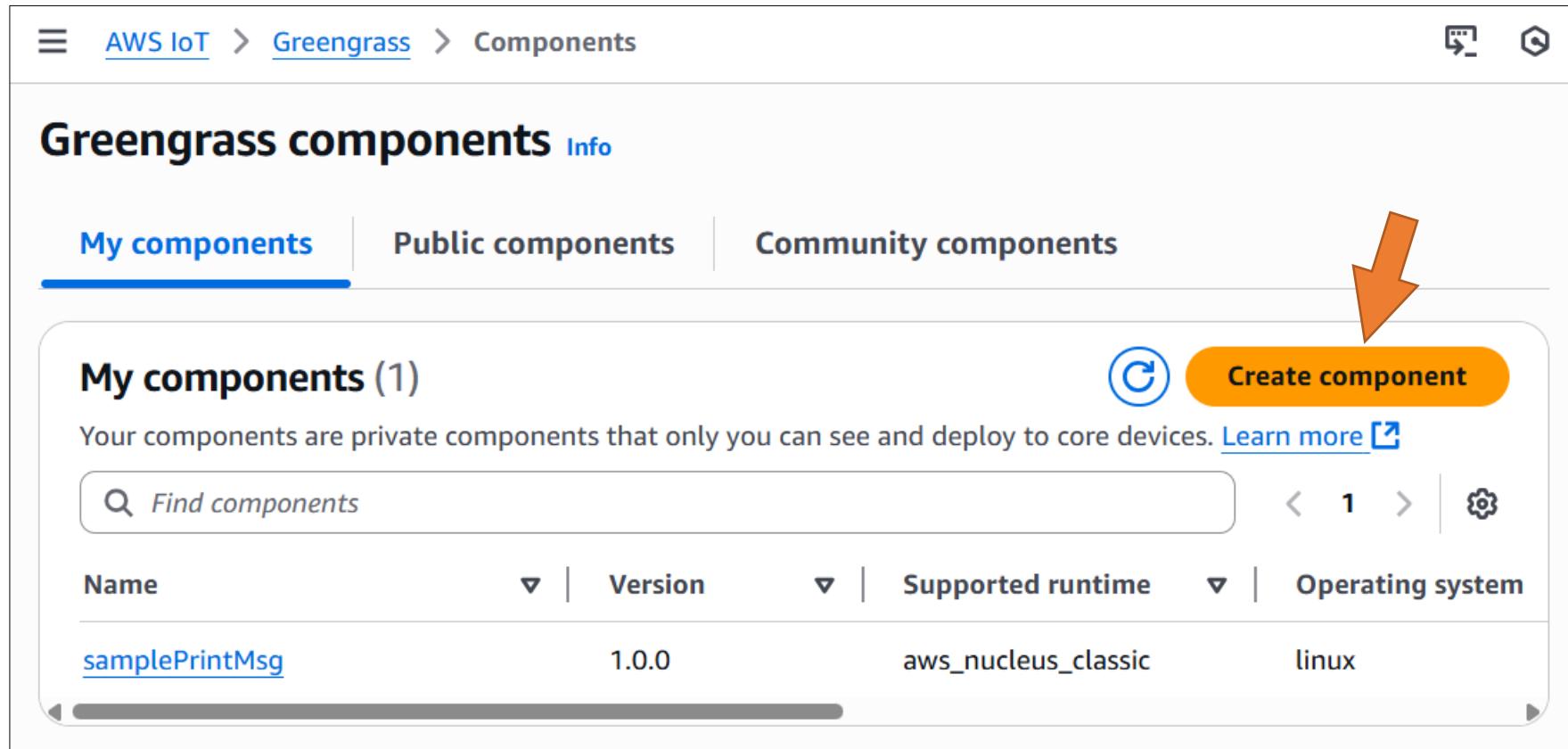
- Upload your artifacts to a folder in your S3 bucket



```
cmpe583-PubSub.py x
S3 > artifacts > cmpe583-PubSub.py
1  from datetime import datetime
2  import time
3  import traceback
4  import json
5  import botocore
6  import sys
7  import os
8
9  import awsiot.greengrasscoreipc
10 import awsiot.greengrasscoreipc.client as client
11 from awsiot.greengrasscoreipc.model import (
12     IoTCoreMessage,
13     QoS,
14     SubscribeToIoTCoreRequest,
15     PublishToIoTCoreRequest
16 )
17
18 TIMEOUT = 10
19 REQUEST_TOPIC = sys.argv[1]
20 RESPONSE_TOPIC = sys.argv[2]
21 THING_NAME = os.getenv('AWS_IOT_THING_NAME')
22
23 ipc_client = awsiot.greengrasscoreipc.connect()
```

# Step 2: Create a Custom Component

## Via AWS IoT Greengrass (Console)



The screenshot shows the AWS IoT Greengrass Components page. The navigation bar at the top includes 'AWS IoT > Greengrass > Components'. Below the navigation, the title 'Greengrass components' has an 'Info' link. There are three tabs: 'My components' (which is selected and underlined), 'Public components', and 'Community components'. An 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 'My components (1)'. It contains a search bar with the placeholder 'Find components', a pagination indicator showing page 1 of 1, and a settings gear icon. A table lists one component: 'samplePrintMsg' (Version 1.0.0, Supported runtime aws\_nucleus\_classic, Operating system linux). The table has columns for Name, Version, Supported runtime, and Operating system.

Name	Version	Supported runtime	Operating system
<a href="#">samplePrintMsg</a>	1.0.0	aws_nucleus_classic	linux

# Step 2: Create a Custom Component

## Enter the Recipe

The screenshot shows the 'Create component' page in the AWS IoT Greengrass console. The navigation bar at the top includes 'AWS IoT > Greengrass > Components > Create component'. The main heading 'Create component' is followed by a sub-instruction: '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.' Below this, the 'Component information' section is titled 'Component information' and describes creating a component from a recipe or importing an AWS Lambda function. An orange arrow points to the 'Enter recipe as JSON' button. This button is selected, indicated by a blue border and a checked radio button. The sub-instruction below it says 'Start with an example or enter your recipe.' To the right are two other options: 'Enter recipe as YAML' (unchecked) and 'Import Lambda function' (unchecked). The 'Recipe' section below contains instructions about artifact availability in an S3 bucket and a 'Learn more' link. A large code editor window displays a JSON component recipe with numbered lines 1 through 10. Lines 1-10 show the following JSON structure:

```
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 response to messages",  
7     "ComponentPublisher": "<Name>",  
8     "ComponentConfiguration": {  
9         "DefaultConfiguration": {  
10            "accessControl": {  
11                "allow": [  
12                    "topic:sampleTopic"  
13                ]  
14            }  
15        }  
16    }  
17}
```

# Step 2: Create a Custom Component

## Adjust the Access Control

```
8 ▼   "ComponentConfiguration": {  
9 ▼     "DefaultConfiguration": {  
10 ▼       "accessControl": {  
11 ▼         "aws.greengrass.ipc.mqttproxy": {  
12 ▼           "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {  
13             "policyDescription": "Allows access to publish/subscribe  
14             "operations": [  
15               "aws.greengrass#PublishToIoTCore",  
16               "aws.greengrass#SubscribeToIoTCore"  
17             ],  
18             "resources": [  
19               "*"  
20             ]  
21           }  
22         }  
23     }  
24 }
```

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 2: Create a Custom Component

## Adjust the Access Control

```
"Manifests": [
    {
        "Lifecycle": {
            "Install": "pip3 install --user awsiotsdk botocore --break-system-packages",
            "Run": "python3 -u {artifacts:path}/cmpe583-PubSub.py cmpe/request cmpe/response\n"
        },
        "Artifacts": [
            {
                "Uri": "s3://cmpe583.greengrass.bucket/artifacts/cmpe583-PubSub.py",
                "Digest": "ZpHa6IOEaf/NST7aso8bf52AP39DNOeNtknC/5Qah/E=",
                "Algorithm": "SHA-256",
                "Unarchive": "NONE",
                "Permission": {
                    "Read": "OWNER",
                    "Execute": "NONE"
                }
            }
        ]
    }
}
```

Make sure python3 and pip3 are available on your core device!

awsiotsdk and botocore libraries are required.

# Step 3: Deploy Your Component

## Via AWS IoT Greengrass (Console)

The screenshot shows the AWS IoT Greengrass Components console. The navigation bar at the top includes 'AWS IoT > Greengrass > Components > samplePubSub:1.0.0'. A green success message box contains the text: 'The component version 1.0.0 was created.' An orange arrow points from this message towards the 'Deploy' button. Below the message, the component name 'samplePubSub' is displayed, along with a dropdown menu showing 'Version: 1.0.0' and buttons for 'Create version', 'Delete version', 'View recipe', and 'Deploy'. The 'Deploy' button is highlighted with an orange border. The main content area is titled 'Overview' and contains sections for 'Description', 'Version', 'Status', 'ARN', 'Publisher', 'Component scope', and 'Type'. The 'Description' section states: 'A component that subscribes to a topic and response to messages'. The 'Version' section shows '1.0.0'. The 'Status' section indicates 'Deployable'. The 'ARN' section provides the full ARN: 'arn:aws:greengrass:eu-central-1:465822364162:components:samplePubSub:versions:1.0.0'. The 'Publisher' section shows '<Name>'. The 'Component scope' section is set to 'Private'. The 'Type' section is 'aws.greengrass.generic'.

The component version 1.0.0 was created.

samplePubSub

Version: 1.0.0 ▾

Create version

Delete version

View recipe

Deploy

### Overview

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

<b>Version</b> 1.0.0	<b>Status</b> Deployable	<b>Publisher</b> <Name>	<b>Type</b> aws.greengrass.generic
<b>ARN</b> arn:aws:greengrass:eu-central-1:465822364162:components:samplePubSub:versions:1.0.0	<b>Version created</b> in 7 seconds	<b>Component scope</b> Private	

# Step 3: Deploy Your Component

## Add Custom Component to Existing Deployment

The screenshot shows the AWS Lambda console interface for a component named "samplePubSub". At the top, there are buttons for "Version: 1.0.0", "Create version", "Delete version", "View recipe", and "Deploy". Below this is a modal dialog titled "Add to deployment". The dialog has two radio button options: "Add to existing deployment" (which is selected and highlighted with a blue border) and "Create new deployment". A search bar labeled "Find by deployment name or target name" is present. Below the search bar is a table with columns: Deployment, Target name, Target type, and Status. One row in the table is selected, showing "Test Deployment for CMPE583-GreenGrassGroup" under Deployment, "CMPE583-GreenGrassGroup" under Target name, "Thing group" under Target type, and "Active" under Status. Navigation arrows for the table are visible on the right. At the bottom of the dialog are "Cancel" and "Next" buttons.

samplePubSub

Version: 1.0.0 ▾ Create version Delete version View recipe Deploy

Add to deployment

Deployment

Add to existing deployment  Create new deployment

Find by deployment name or target name

Deployment	Target name	Target type	Status
<a href="#">Test Deployment for CMPE583-GreenGrassGroup</a>	CMPE583-GreenGrassGroup	Thing group	Active

Cancel Next

# Step 3: Deploy Your Component

## Select the Component

AWS IoT > Greengrass > Deployments > d3379dab-9d13-4bf9-8f2d-b1cd1b25a21c > 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 (2) Info

Find components  Show only selected components < 1 >

Name	Supported runtime	Operating system
samplePubSub	aws_nucleus_classic	All
samplePrintMsg	aws_nucleus_classic	linux



# Step 3: Deploy the Component

## Review & Deploy

AWS IoT > Greengrass > Deployments > f0bcd905-6333-4c55-858c-31521f022bdb > 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

### Review

#### Step 1: Deployment info

**Edit**

#### Deployment target

**Name**  
Test Deployment for CMPE583-GreenGrassGroup

**Target type**  
Thing group

**Target name**  
CMPE583-GreenGrassGroup

Download as JSON

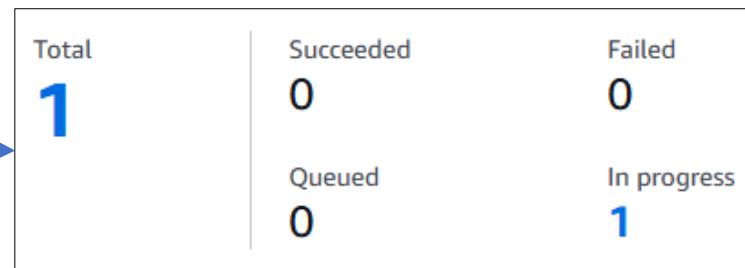
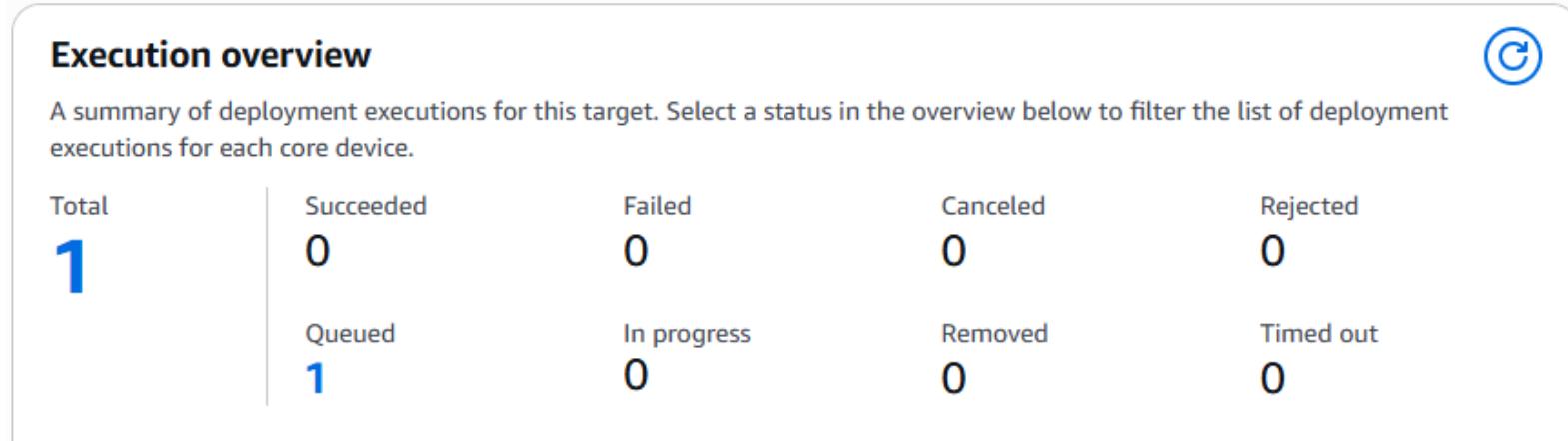
Cancel

Previous

Deploy

# Step 4: Verify the Component

Check AWS IoT Greengrass Console



# Troubleshooting

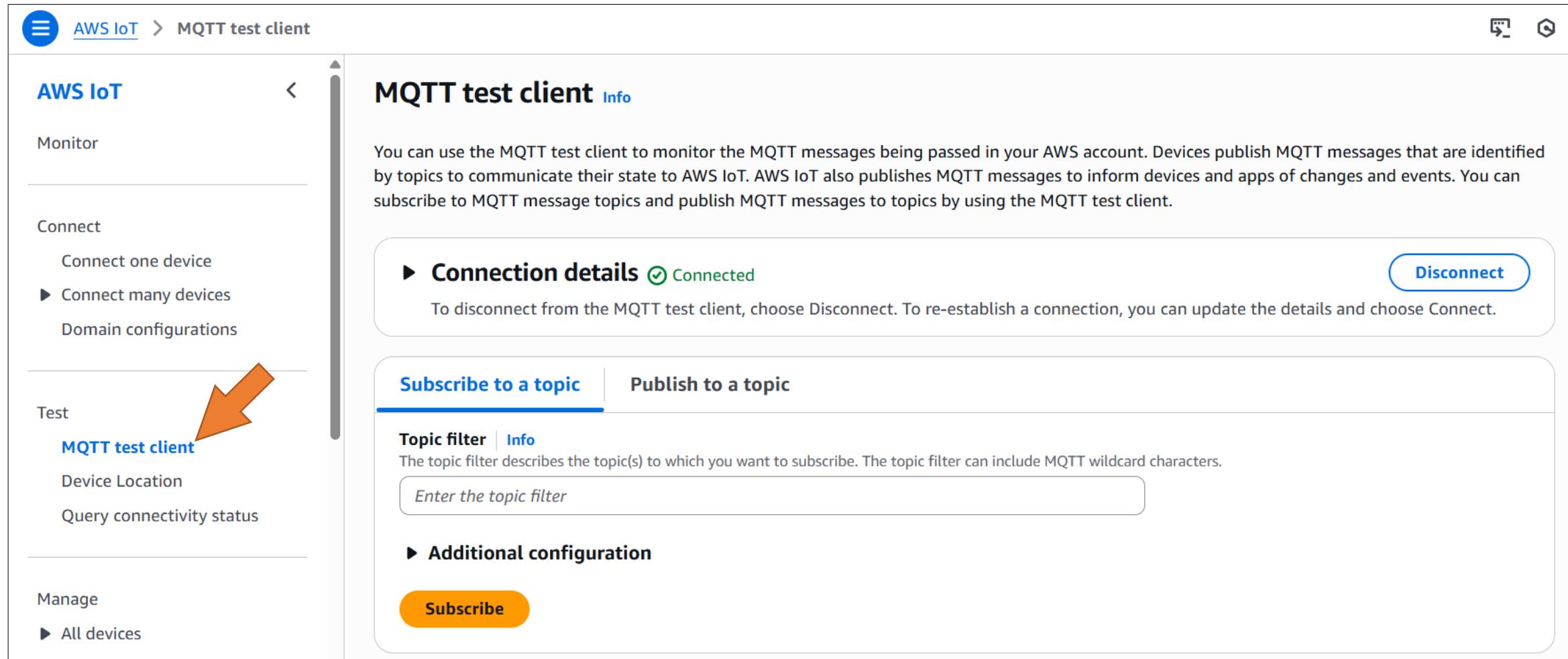
## Check greengrass.log File

- 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 'Test' section with a 'MQTT test client' link, which is highlighted with a large orange arrow pointing to it. The main content area shows 'Connection details' with a 'Connected' status and a 'Disconnect' button. It also includes sections for 'Subscribe to a topic' (with a 'Topic filter' input field) and 'Publish to a topic'. At the bottom, there's an 'Additional configuration' section with a 'Subscribe' button.

AWS IoT > MQTT test client

**AWS IoT**

- Monitor
- Connect
  - Connect one device
  - ▶ Connect many devices
  - Domain configurations
- Test
  - MQTT test client**
  - Device Location
  - Query connectivity status
- Manage
  - ▶ All devices

**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 Disconnect

To disconnect from the MQTT test client, choose Disconnect. To re-establish a connection, you can update the details and choose Connect.

**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 deployment. At the top, there are two tabs: "Subscribe to a topic" (which is active, indicated by a blue underline) 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 link next to the topic filter.
- Description:** Text explaining that the topic filter describes the topic(s) to which you want to subscribe, mentioning MQTT wildcard characters.
- Additional configuration:** A section with a disclosure triangle icon.
- Subscribe button:** An orange button.

**Publish to a topic:**

- Subscriptions:** A table with one row showing the topic "#".
- Action buttons:** "Pause", "Clear", "Export", and "Edit".
- Message delivery info:** A callout box stating: "You cannot publish messages to a wildcard topic. Please select a different topic to publish messages to."

# Test Your Deployment

## Send a Test Message

The screenshot shows the AWS IoT Publish to a topic interface. At the top, there are two buttons: "Subscribe to a topic" and "Publish to a topic", with "Publish to a topic" being the active one. Below the buttons, there is a section titled "Topic name" with a descriptive text: "The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0." An orange arrow points to the "Topic name" input field, which contains the value "cmpe/request". To the right of the input field is a blue "X" button. Below the topic name section is a "Message payload" section containing a JSON object: { "message": "Test message from AWS IoT console" }. At the bottom of the interface 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.

X

**Message payload**

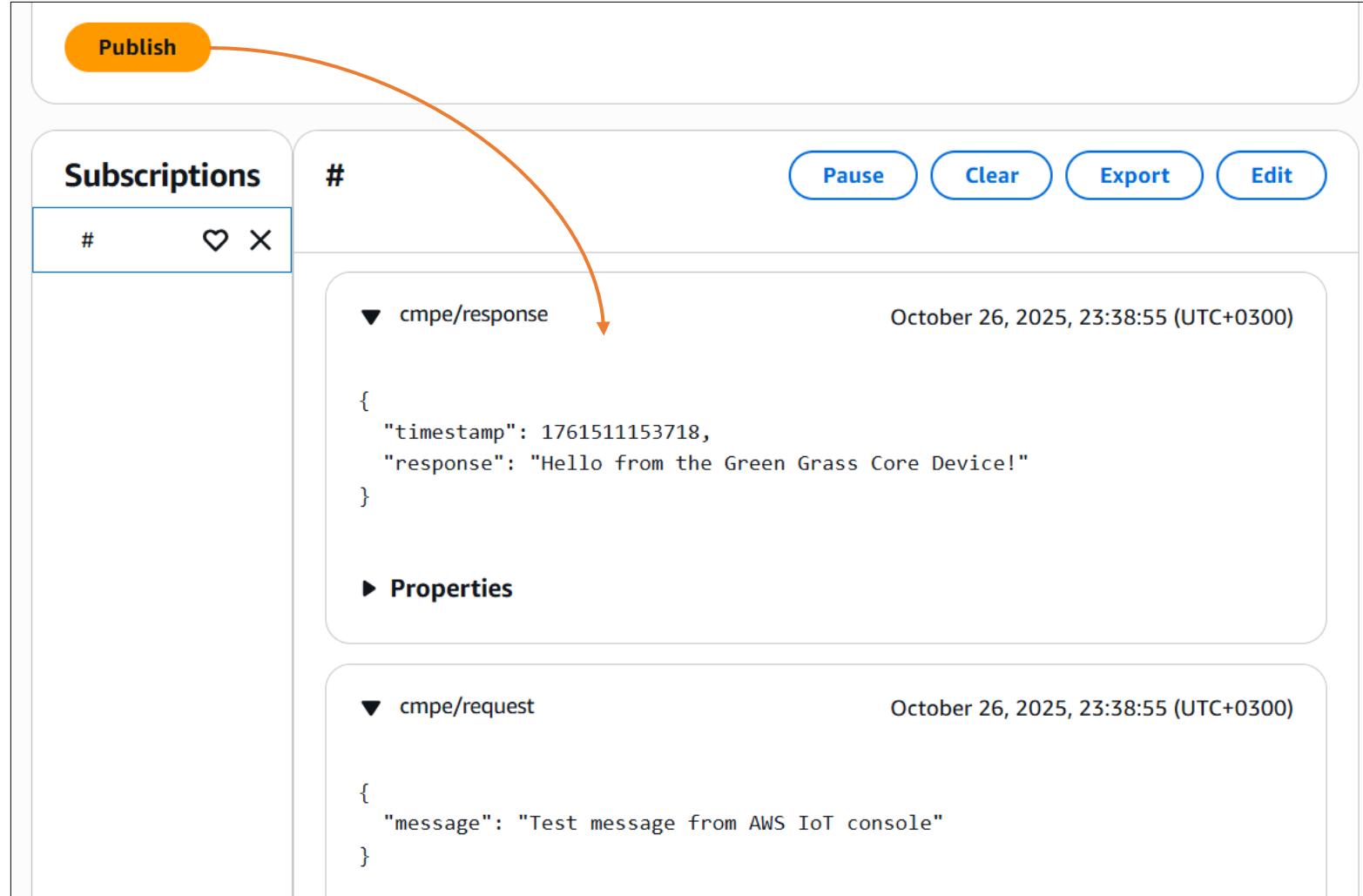
```
{  
  "message": "Test message from AWS IoT console"  
}
```

▶ Additional configuration

**Publish**

# Test Your Deployment

## Check If the Client Responds Back



# Other Use-Cases of AWS IoT Greengrass Core IPC

- Interact with component lifecycle (component-to-component or component-to-cloud).
  - Pause component, resume component
  - Listen to lifecycle events (start, stop, etc.)
- Interact with component configuration.
  - Get and set component configuration parameters
- Retrieve secret values (access AWS Secrets Manager secrets securely).
  - 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.

# On-demand vs Long-lived Functions

Feature	On-demand Function	Long-lived (Pinned) Function
Lifecycle Trigger	Starts only when invoked by an event (e.g., an MQTT message or an IPC call).	Starts automatically with the Greengrass Core component and runs continuously.
Execution Duration	<b>Short-lived.</b> The function's container is shut down after a period of inactivity.	<b>Persistent.</b> Runs indefinitely. If the function fails, Greengrass automatically restarts it.
State Management	Primarily <b>Stateless</b> . Cannot reliably maintain state in memory between invocations.	Primarily <b>Stateful</b> . Ideal for maintaining state, open connections, and managing in-memory caches.
Resource Usage	<b>Low / Intermittent.</b> Consumes CPU and memory only when active, making it efficient for resource-constrained devices.	<b>High / Constant.</b> Consumes resources continuously as long as the Greengrass Core is running.
Configuration	Configured with <b>Pinned: false</b> (default).	Configured with <b>Pinned: true</b> .
Ideal Use Cases	<ul style="list-style-type: none"><li>• Responding to sensor data triggers</li><li>• Message transformation and routing</li><li>• Simple, event-driven logic that runs infrequently</li></ul>	<ul style="list-style-type: none"><li>• Continuous data acquisition from local devices</li><li>• Running a local ML inference model.</li><li>• Managing a persistent connection to a local database or device.</li></ul>

# Steps for Deploying a Component Using Lambda

1

Create an AWS Lambda function and publish it

2

Create a custom component using the Lambda function

3

Deploy your Greengrass component and define a trigger function

4

Verify and test the Greengrass component

# Step 1: Create AWS Lambda Function

## Via AWS Lambda (Console)

The screenshot shows the AWS Lambda Functions console with 5 functions listed. A prominent orange arrow points from the 'Create function' button in the top-left corner of the main interface to the 'Create function' step in the 'Create function' wizard.

**Potential Issues:**  
**Runtime Version Mismatch**  
Try to use same Runtime and Architecture on AWS and your Core Device! Otherwise use [FunctionRuntimeOverride](#) in your Greengrass configuration

**Create function** Info  
Choose one of the following options to create your function.

- Author from scratch  
Start with a simple Hello World example.
- Use a blueprint  
Build a Lambda application from sample code and configuration presets for common use cases.
- Container image  
Select a container image to deploy for your function.

**Basic information**

**Function name**  
Enter a name that describes the purpose of your function.

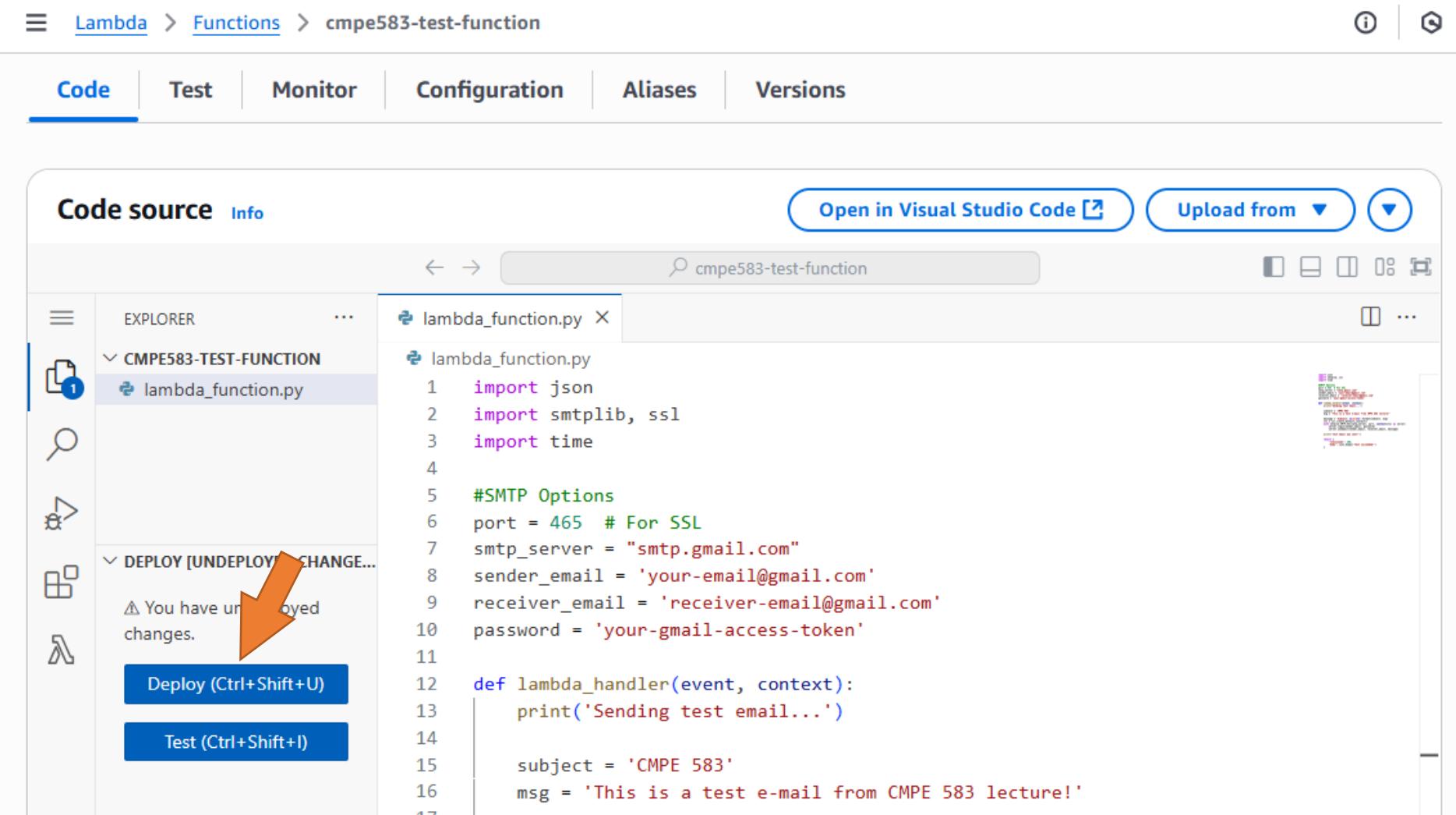
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

**Runtime** Info  
Choose the language you use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

**Architecture** Info  
Choose the instruction set architecture you want for your function code.  
 arm64  
 x86\_64

# Step 1: Create AWS Lambda Function

## Deploy the Lambda Function



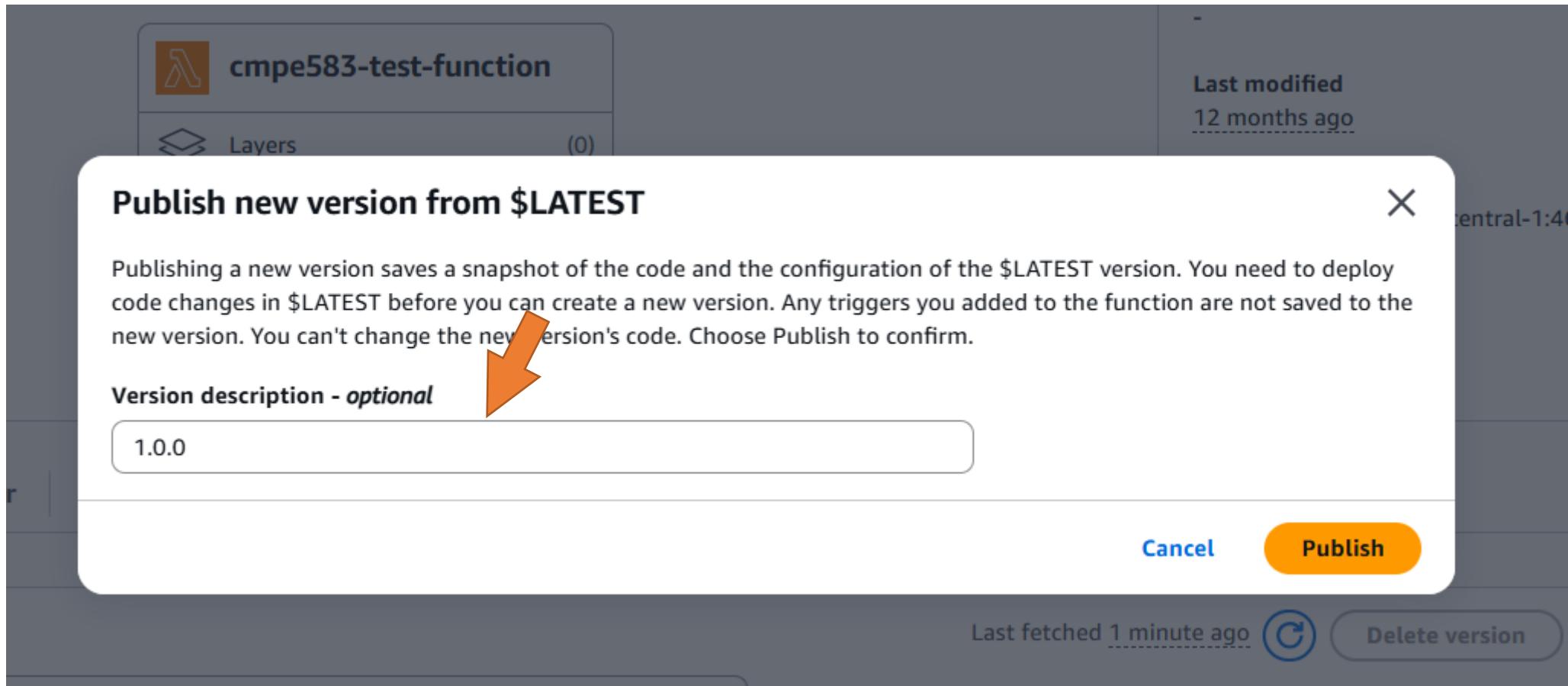
The screenshot shows the AWS Lambda Functions console with the path `Lambda > Functions > cmpe583-test-function`. The `Code` tab is selected. In the `Code source` section, there is an `EXPLORER` sidebar showing a folder named `CMPE583-TEST-FUNCTION` containing a file named `lambda_function.py`. The main area displays the code for `lambda_function.py`:

```
1 import json
2 import smtplib, ssl
3 import time
4
5 #SMTP Options
6 port = 465 # For SSL
7 smtp_server = "smtp.gmail.com"
8 sender_email = 'your-email@gmail.com'
9 receiver_email = 'receiver-email@gmail.com'
10 password = 'your-gmail-access-token'
11
12 def lambda_handler(event, context):
13     print('Sending test email...')
14
15     subject = 'CMPE 583'
16     msg = 'This is a test e-mail from CMPE 583 lecture!'
```

An orange arrow points to the `Deploy (Ctrl+Shift+U)` button in the `EXPLORER` sidebar. The sidebar also displays a message: `⚠ You have un-deployed changes.`

# Step 1: Create AWS Lambda Function

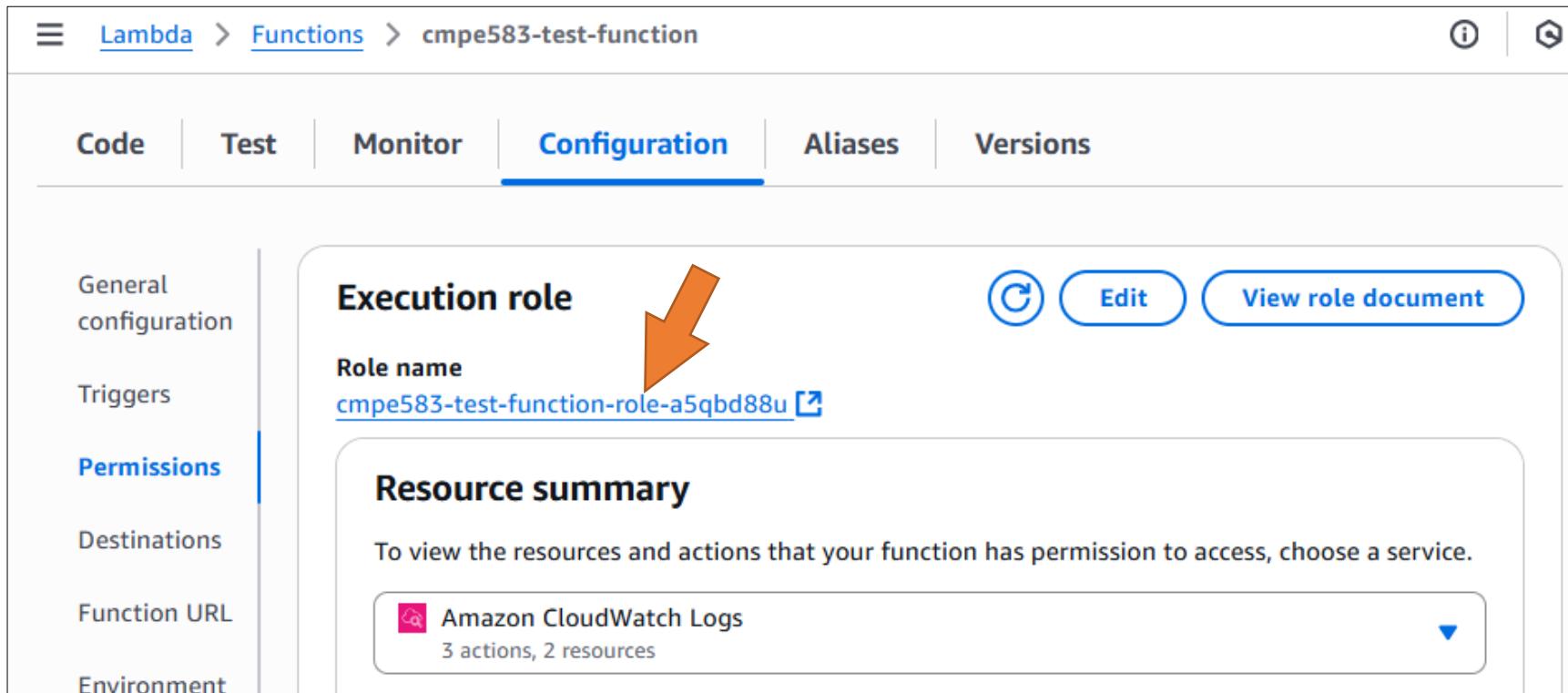
## Publish the First Version



# Step 1: Create AWS Lambda Function

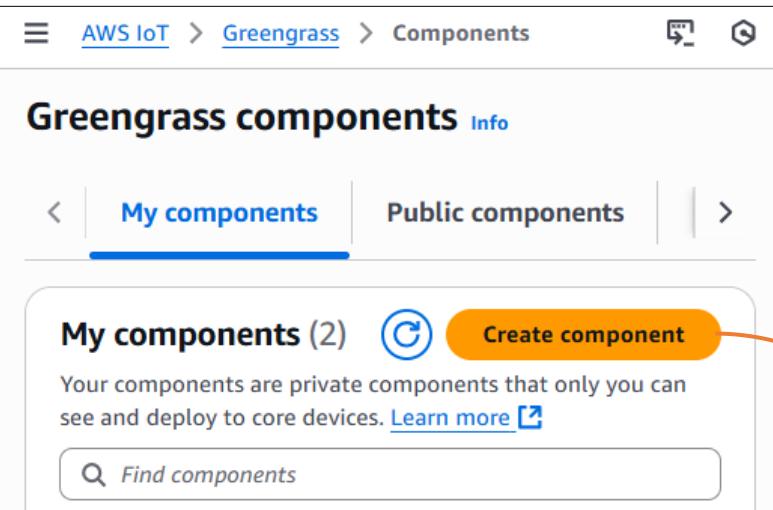
## Configure Lambda Role

- 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.

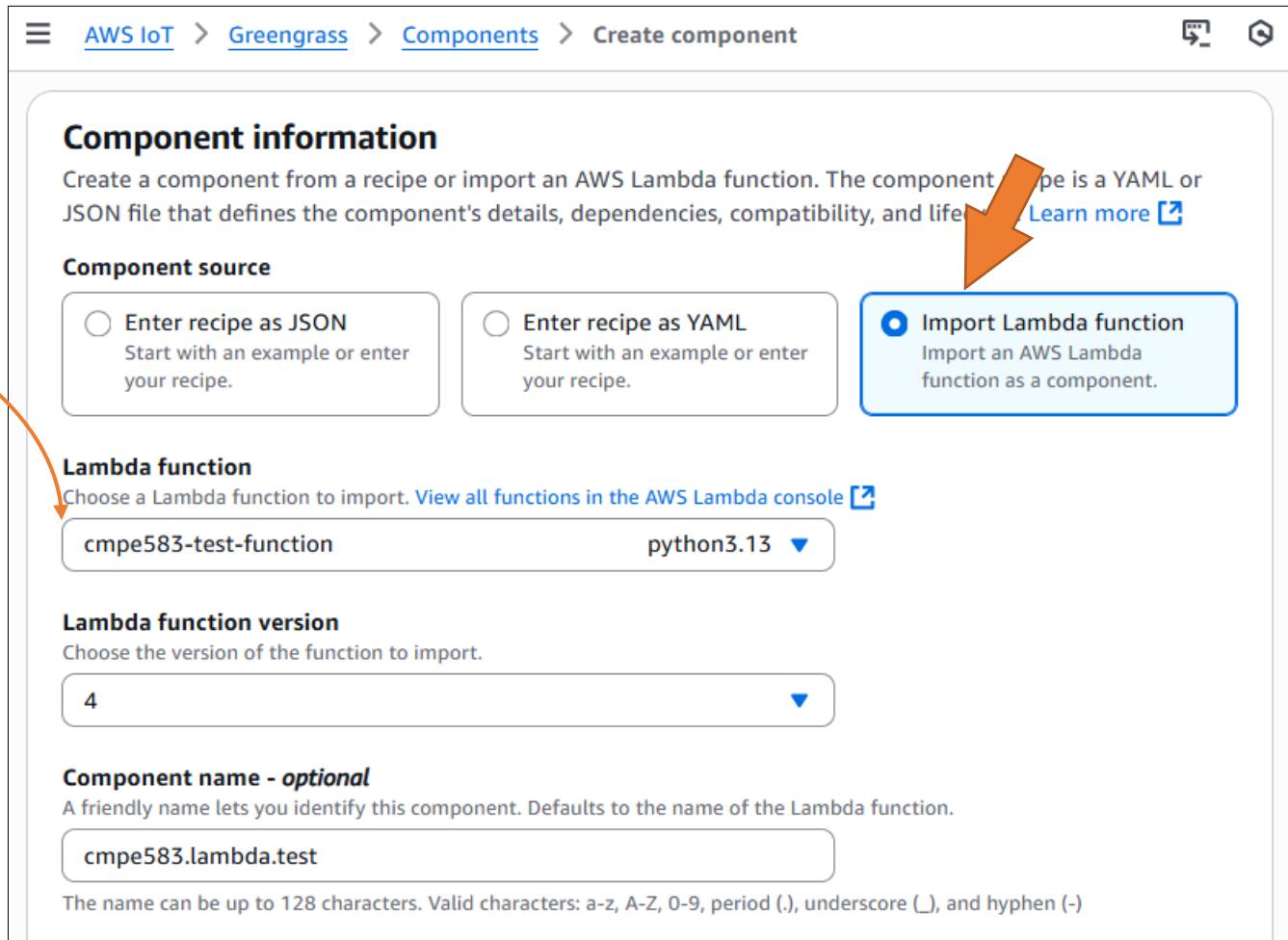


# Step 2: Create a Custom Component

## Import Lambda Function



The screenshot shows the AWS Greengrass Components page. At the top, there are navigation links: AWS IoT > Greengrass > Components. Below this, the title "Greengrass components" is followed by a "Info" link. There are two tabs: "My components" (which is selected and highlighted in blue) and "Public components". Under the "My components" tab, it says "My components (2)" and there is a "Create component" button with a circular icon containing a 'C'. A red arrow points from this button to the "Import Lambda function" section on the right-hand page.



The screenshot shows the "Create component" wizard. The title is "Component information". It explains that a component can be created from a recipe or imported as a Lambda function. A "Learn more" link is available. The "Component source" section has three options: "Enter recipe as JSON", "Enter recipe as YAML", and "Import Lambda function" (which is selected and highlighted in blue). A red arrow points to this selected option. Below this, the "Lambda function" section shows "cmpe583-test-function" and "python3.13" (with a dropdown arrow). The "Lambda function version" section shows a dropdown menu with the value "4". The "Component name - optional" section shows the input field "cmpe583.lambda.test". A note at the bottom states: "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 Function) and Lifecycle

**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 this component for messages. The component can act on local publish/subscribe messages and AWS IoT Core MQTT messages.

**Topic**  **Type**  [Remove](#)

[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.

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**

**Linux process configuration - optional**

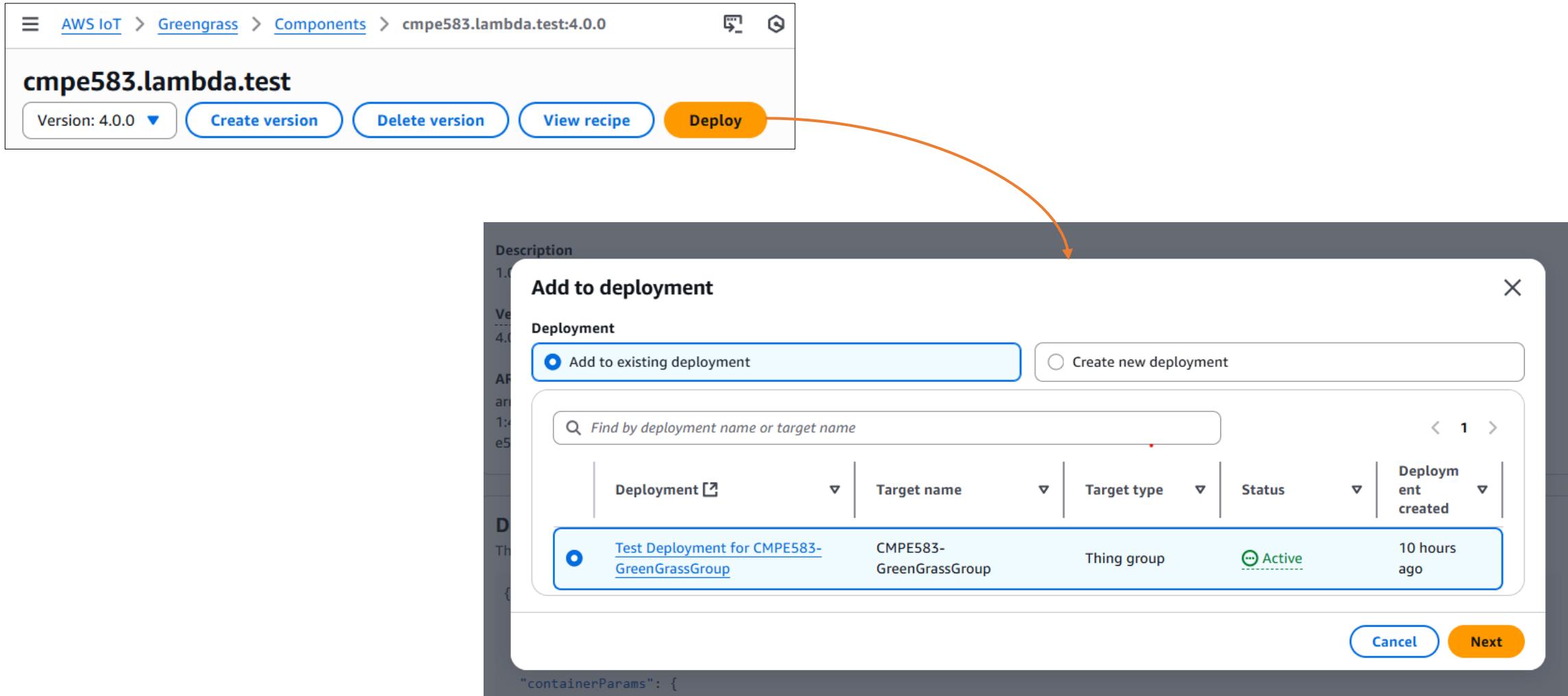
You can define default configuration options for the Linux process that runs this component. When you deploy this component you can override these default values.

**Isolation mode**

Run the process in an isolated container in the AWS IoT Greengrass Core software or as a regular process without isolation.

# Step 3: Deploy Your Component

## Add Component to Existing Deployment



# 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 (5)**

Name	Version	Supported runtime	Modified?
cmpe583.lambda.test	4.0.0	aws_nucleus_classic	-
samplePrintMsg	1.0.0	aws_nucleus_classic	-
samplePubSub	3.0.0	aws_nucleus_classic	-
aws.greengrass.Cli	2.15.0	aws_nucleus_classic	-
aws.greengrass.client_devices.IPDetector	2.2.2	aws_nucleus_classic	-

**Configure component**

**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 and response...",  
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...",  
14                        "operations": [  
15                            "aws.greengrass#PublishToIoTCore",  
16                            "aws.greengrass#SubscribeToIoTCore"  
17                        ],  
18                    },  
19                    "resources": [  
20                        "cmpe/lambda"  
21                    ]  
22                }  
23            }  
24        }  
25    }  
26}
```

**JSON** Ln 26, Col 28 Errors: 0 Warnings: 0

# Step 3: Deploy the Component

## Review & Deploy

AWS IoT > Greengrass > Deployments > f0bcd905-6333-4c55-858c-31521f022bdb > 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

### Review

#### Step 1: Deployment info

**Edit**

#### Deployment target

**Name**  
Test Deployment for CMPE583-GreenGrassGroup

**Target type**  
Thing group

**Target name**  
CMPE583-GreenGrassGroup

Download as JSON

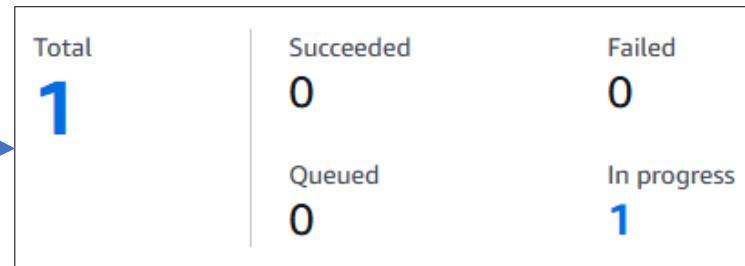
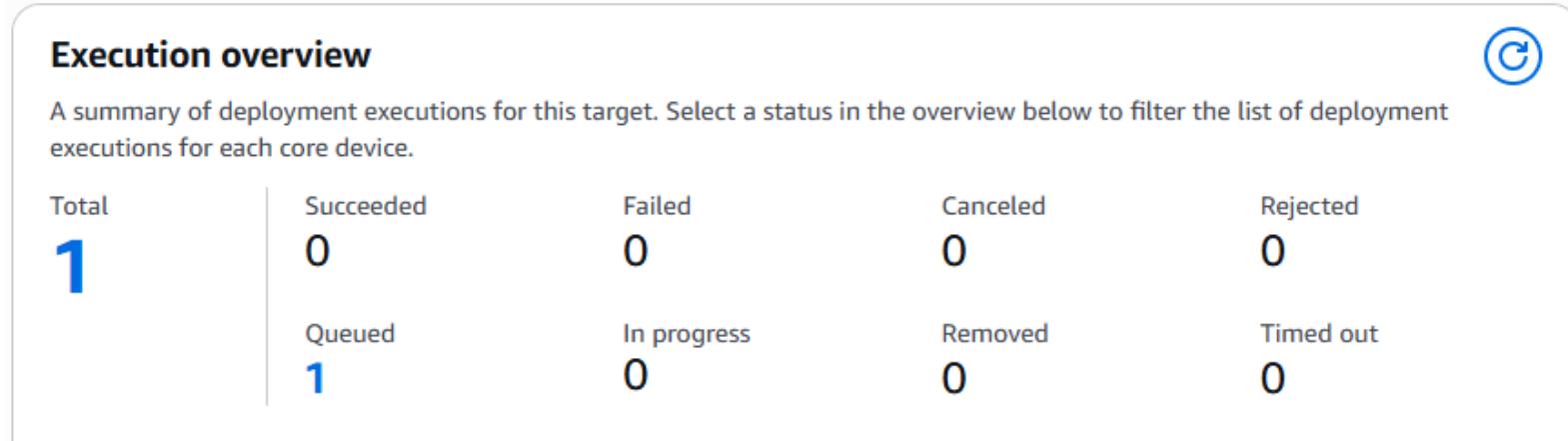
Cancel

Previous

Deploy

# Step 4: Verify the Component

Check AWS IoT Greengrass Console



# Test Your Deployment

## Via AWS 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 [Disconnect](#)

To disconnect from the MQTT test client, choose Disconnect. To re-establish a connection, you can update the details and choose Connect.

**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.

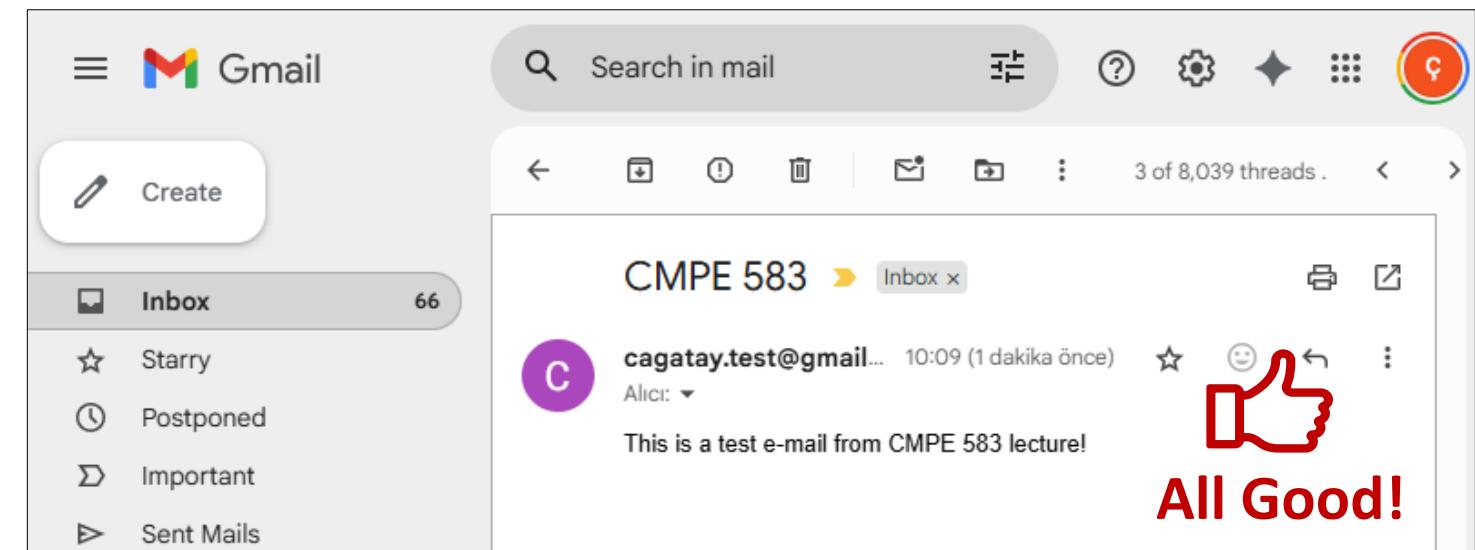
cmpe/lambda [X](#)

**Message payload**

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

► **Additional configuration**

[Publish](#)



# 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}
```



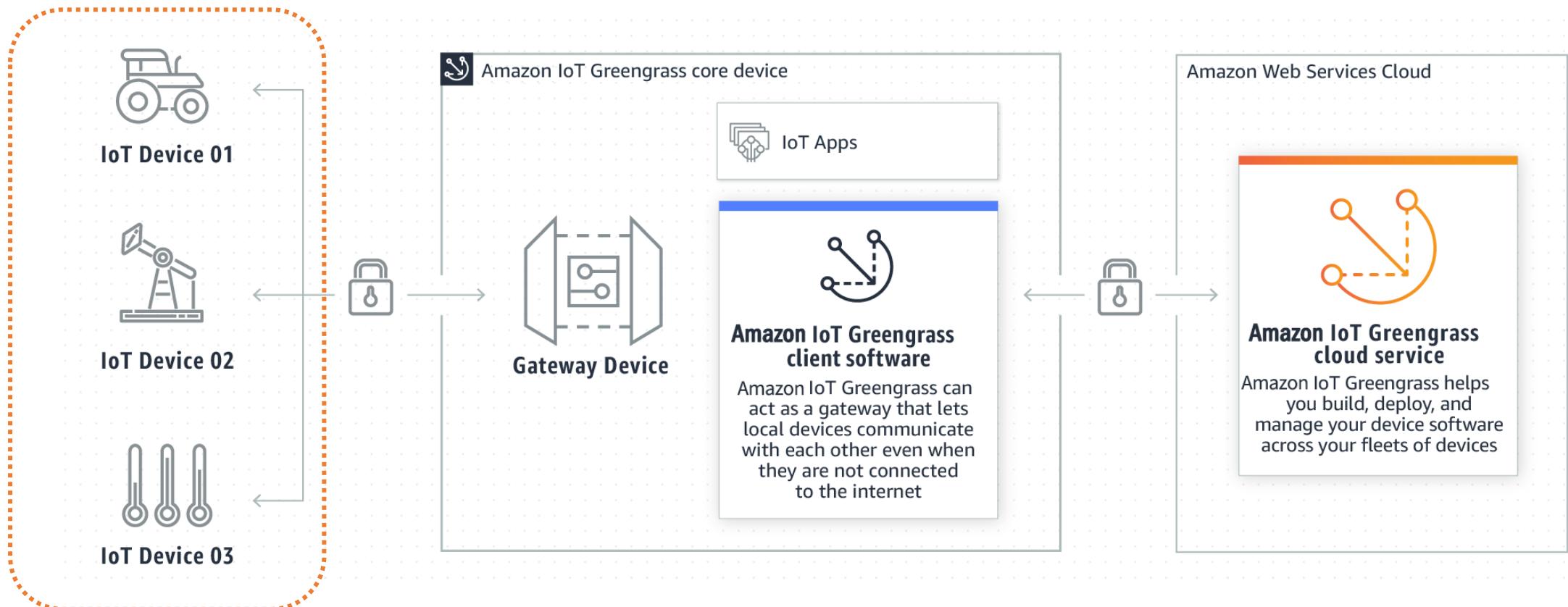
# Edge Computing with AWS IoT Greengrass - Part #2

# Interact with Local IoT Devices

---

# Local IoT Devices

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



# Connect Client Devices to Core Devices

- You must configure cloud discovery to connect client devices to **core devices**.
- Associate client devices with a core device so that they can discover the core device.
- Use the AWS IoT Device SDK on **client devices** to discover, connect, and communicate with a core device.
- You can perform both local (edge) and remote (Cloud) operations:
  - Local (Edge) Operations:
    - Local MQTT messaging with Greengrass components and client devices.
    - Interact with client device (local) shadows in Greengrass components.
  - Remote (Cloud) Operations:
    - Relay MQTT messages between client devices and AWS IoT Core.
    - Sync client devices (remote) shadows with AWS IoT Core.

# AWS-Provided Client Device Components

- AWS provides the following public components that you can deploy to **core devices**.
- These components enable **client devices** to connect & communicate with core devices.

Component	Description	Component type	Supported OS	Open source
Client device auth	Enables local IoT devices, called client devices, to connect to the core device.	Plugin	Linux, Windows	Yes
IP detector	Reports MQTT broker connectivity information to AWS IoT Greengrass, so client devices can discover how to connect.	Plugin	Linux, Windows	Yes
MQTT bridge	Relays MQTT messages between client devices, local AWS IoT Greengrass publish/subscribe, and AWS IoT Core.	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.	Plugin	Linux, Windows	Yes
MQTT 5 broker (EMQX)	Runs an MQTT 5 broker that handles messages between client devices and the core device.	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.	Plugin	Linux, Windows	Yes

# Creating Client Devices

---

# Steps for Creating a Client Device

1

Configure the  
Greengrass  
Service Role

2

Configure the  
AWS IoT Thing  
Policy

3

Create the Client  
Device (Thing)  
via AWS IoT  
Console

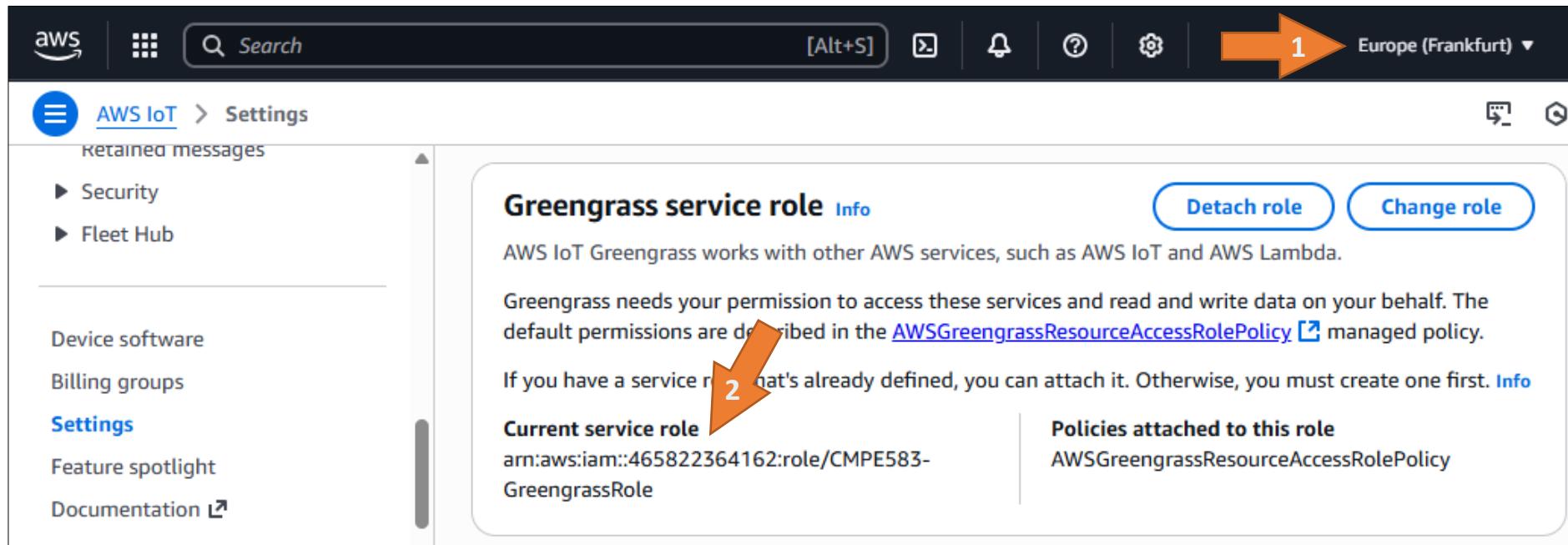
4

Download  
Certificates and  
Keys (needed for  
secure auth. and  
communication)

# 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.



# Step 2: Configure the AWS IoT Thing Policy

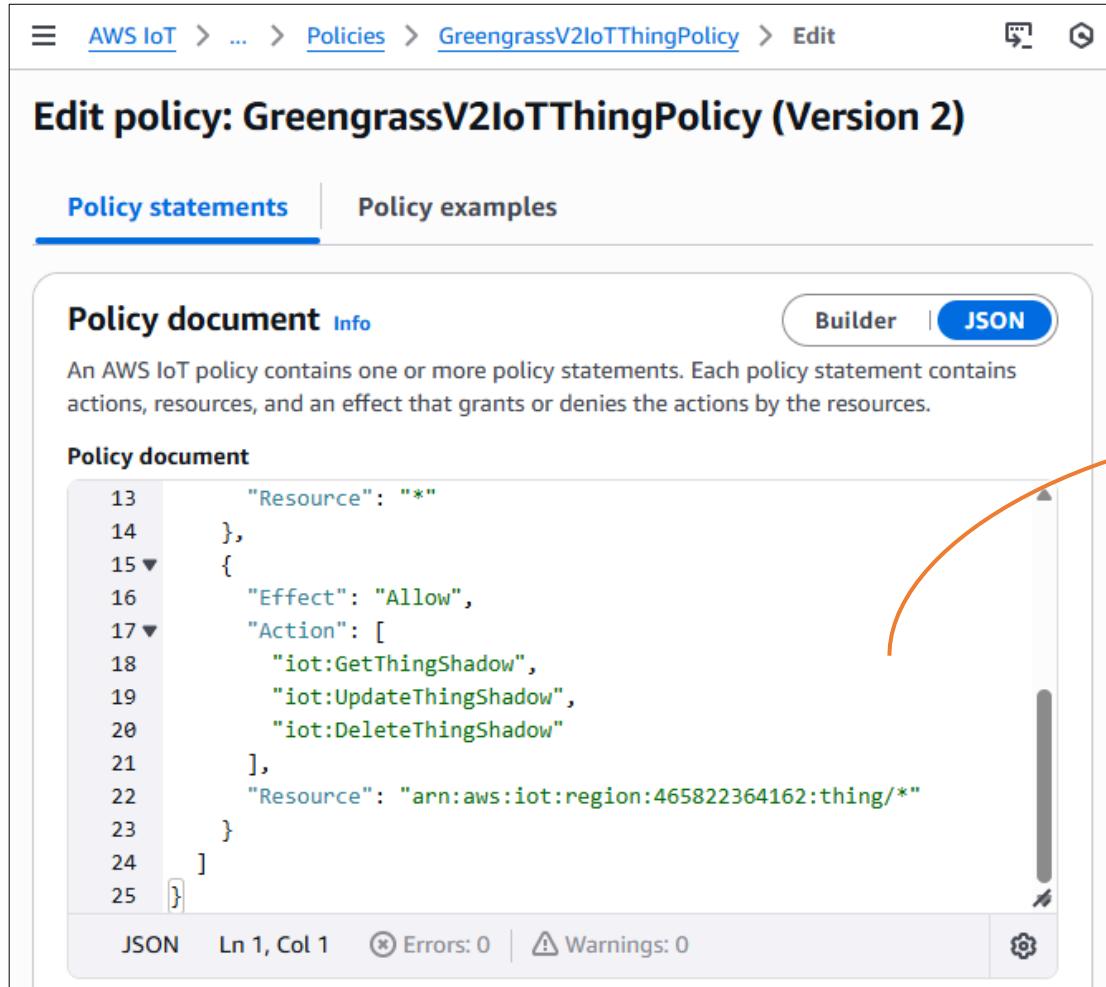
## Review the Policy

The image shows two screenshots of the AWS IoT console. The left screenshot displays the 'AWS IoT policies' page with four policies listed: 'MyOTPPolicyVeryExtensive2', 'MyFirstIoTThing-Policy', 'GreengrassV2IoTThingPolicy', and 'GreengrassTESCertificatePolicyGreengrassV2TokenExchangeRoleAlias'. An orange arrow points from the 'GreengrassV2IoTThingPolicy' link on this page to the right screenshot. The right screenshot shows the details for the 'GreengrassV2IoTThingPolicy'. It includes fields for 'Policy ARN' (arn:aws:iot:eu-central-1:465822364162:policy/GreengrassV2IoTThingPolicy), 'Active version' (2), 'Created' (September 19, 2023, 18:55:56 (UTC+03:00)), and 'Last updated' (September 19, 2023, 18:55:56 (UTC+03:00)). A large orange arrow points to the 'Edit active version' button at the top right of the details page.

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

# Step 2: Configure the AWS IoT Thing Policy

Optional: Allow Core Device to Sync Shadows



The screenshot shows the AWS IoT Policies page with the path: AWS IoT > Policies > GreengrassV2IoTThingPolicy > Edit. The title is "Edit policy: GreengrassV2IoTThingPolicy (Version 2)". There are two tabs: "Policy statements" (selected) and "Policy examples". The "Policy document" section contains the following JSON:

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

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.

The "JSON" tab is selected at the bottom left, showing "Ln 1, Col 1" and "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 shows the AWS IoT Policies configuration screen. On the left, a modal dialog titled "Policy version status" is open, containing the following text:

**Active policy**  
 Set the edited version as the active version for this policy  
You can change this setting later in the policy's detail page.

At the bottom of this dialog are two buttons: "Cancel" and "Save as new version". An orange arrow points from the "Save as new version" button to the "Save as new version" button on the main policy page.

On the right, the main policy page displays the "Active version: 2" table. The table has three columns: "Policy effect", "Policy action", and "Policy resource". The rows are:

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/*

A callout box with a dashed orange border highlights the last three rows of the table, which correspond to the Device Shadow service permissions. A grey arrow points from this callout box to a note below.

**Security Best Practice: Principle of Least Privilege**  
Only add these permissions if your application specifically uses the Device Shadow service.

# Step 3: Create Client Device (Thing)

Via AWS IoT Console

The screenshot shows the AWS IoT Things management interface. On the left, a sidebar lists various device categories: MQTT test client, Device Location, Query connectivity status, Manage, All devices (with Things selected), Thing groups, Thing types, Fleet metrics, Greengrass devices, LPWAN devices, and Software packages. An orange arrow points to the 'Things' link under 'All devices'. On the right, the main panel displays 'Things (1) Info' with a summary: 'An IoT thing is a representation and record of your physical device in the cloud. A physical device needs a thing record in order to work with AWS IoT.' Below this is a search bar with the placeholder 'Filter things by: name, type, group, billing, or searchable attribute.' and a pagination section with page 1 of 1. A table lists one item: CMPE583-GreenGrassCore. The 'Create things' button at the top right is also highlighted with an orange arrow.

# Step 3: Create Client Device (Thing)

Via AWS IoT Console

The image shows two screenshots of the AWS IoT 'Create things' wizard. The left screenshot shows the 'Number of things to create' step, where 'Create single thing' is selected. The right screenshot shows the 'Specify thing properties' step, where the 'Thing name' is set to 'CMPE\_Test\_Client1'. A red arrow points from the 'Next' button in the first screenshot to the 'Thing name' field in the second screenshot.

**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**

**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.

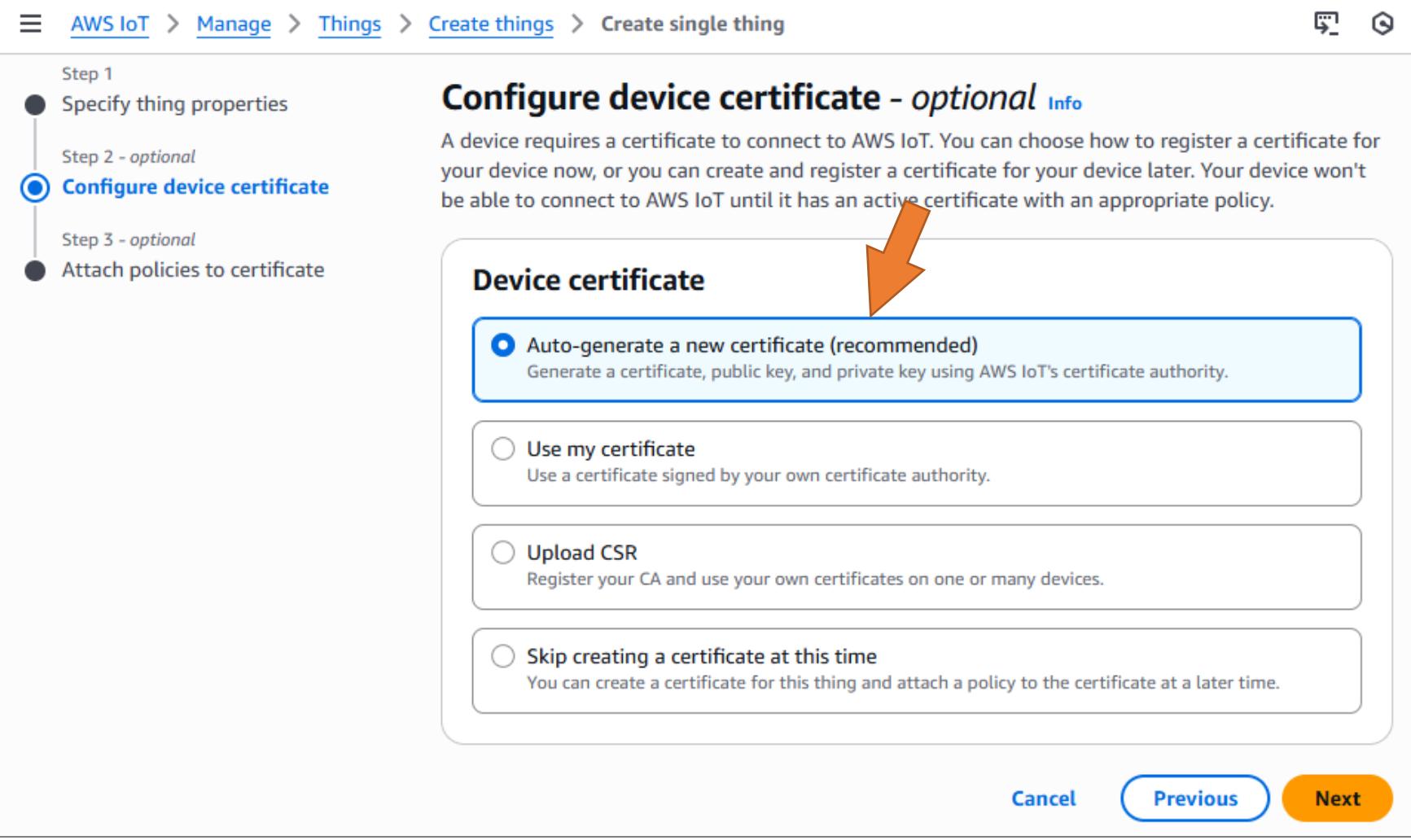
**Thing properties Info**

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

**Additional configurations**  
You can use these configurations to add detail that can help you to organize, manage, and search your things.

# Step 3: Create Client Device (Thing)

## Configure Device Certificate



The screenshot shows the AWS IoT 'Create single thing' wizard at Step 3: Configure device certificate. The navigation path is: AWS IoT > Manage > Things > Create things > Create single thing. The left sidebar lists steps: Step 1 (Specify thing properties), Step 2 - optional (Configure device certificate, which is selected), and Step 3 - optional (Attach policies to certificate). The main content area is titled 'Configure device certificate - optional'. It explains that a device requires a certificate to connect to AWS IoT and provides options for generating or uploading a certificate. A large orange arrow points to the 'Auto-generate a new certificate (recommended)' option.

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

Step 1  
Specify thing properties

Step 2 - optional  
**Configure device certificate**

Step 3 - optional  
Attach policies to certificate

**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.

Use my certificate  
Use a certificate signed by your own certificate authority.

Upload CSR  
Register your CA and use your own certificates on one or many devices.

Skip creating a certificate at this time  
You can create a certificate for this thing and attach a policy to the certificate at a later time.

Cancel Previous Next

# Step 3: Create Client Device (Thing)

## Configure Policy

The screenshot shows the AWS IoT 'Create single thing' wizard at Step 3: Attach policies to certificate. The left sidebar lists three steps: Step 1 (Specify thing properties), Step 2 - optional (Configure device certificate), and Step 3 - optional (Attach policies to certificate). The third step is currently selected and highlighted in blue.

**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

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

An orange arrow points to the checkbox for 'GreengrassV2IoTThingPolicy', which is checked. The interface includes a 'Create policy' button and navigation buttons for 'Cancel', 'Previous', and 'Create thing'.

# Step 4: Download Certificates and Keys

**Download certificates and keys**

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

**Download certificates and keys**

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

Device certificate  
815382b30e1...te.pem.crt

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

**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  
815382b30e1335ef38f4633...8918bd9-public.pem.key

[Download](#) Key downloaded

Private key file  
815382b30e1335ef38f4633...918bd9-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](#) Certificate downloaded

Amazon trust services endpoint  
ECC 256 bit key: Amazon Root CA 3

[Download](#)

If you don't see the root CA certificate that you need here, AWS IoT supports additional root CA certificates. These root CA certificates and others are available in our developer guides. [Learn more ↗](#)

[Download all](#) [Done](#)

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

AWS IoT > Manage > Things

Things (2) [Info](#)

[Advanced search](#) [Run aggregations](#)

An IoT thing is a representation and record of your physical device needs a thing record in order to work with .

Name

CMPE\_Test\_Client1

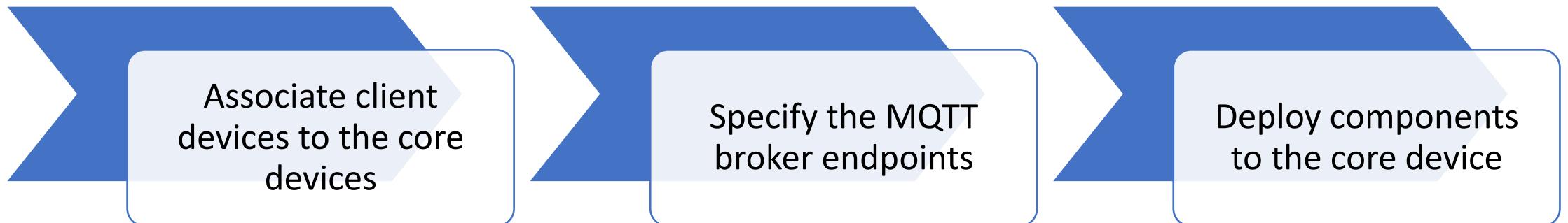
CMPE583-GreenGrassCore

# Configuring Cloud Discovery

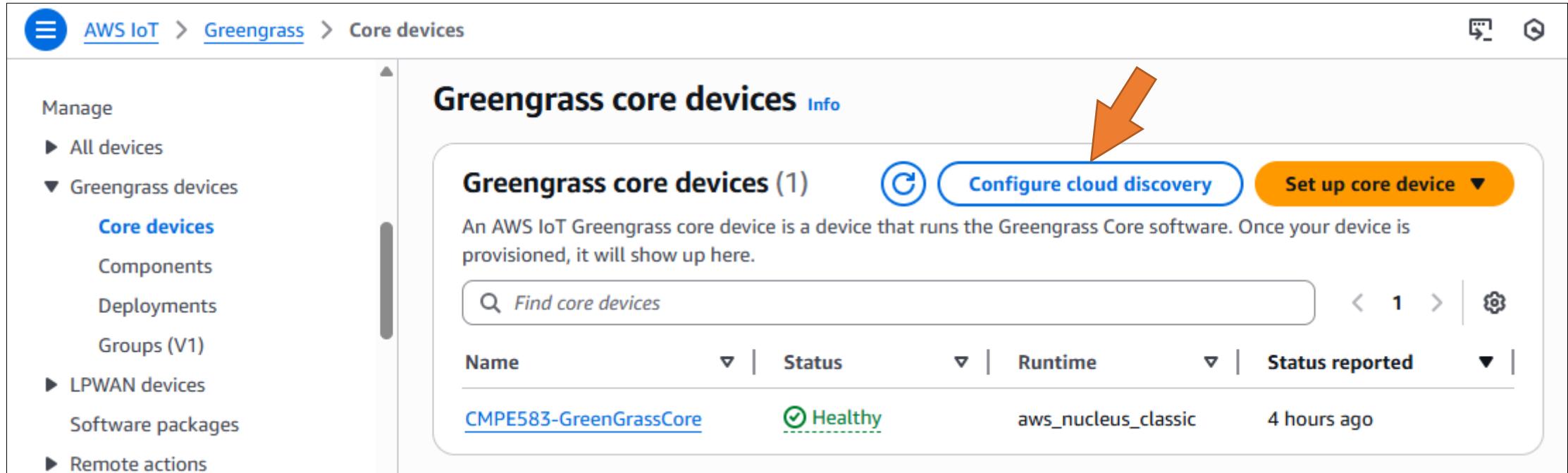
---

# 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:



# Step 1: Configure Cloud Discovery



The screenshot shows the AWS IoT Greengrass Core devices interface. The left sidebar has a 'Manage' section with links: All devices, Greengrass devices (Core devices is selected), Components, Deployments, Groups (V1), LPWAN devices, Software packages, and Remote actions. The main area is titled 'Greengrass core devices' with a 'Info' link. It displays 'Greengrass core devices (1)'. A blue 'Configure cloud discovery' button is highlighted with an orange arrow. Below it is a description: 'An AWS IoT Greengrass core device is a device that runs the Greengrass Core software. Once your device is provisioned, it will show up here.' There is a search bar with 'Find core devices' placeholder text, and navigation controls < 1 >. A table below shows device details: Name (CMPE583-GreenGrassCore), Status (Healthy), Runtime (aws\_nucleus\_classic), and Status reported (4 hours ago).

Name	Status	Runtime	Status reported
CMPE583-GreenGrassCore	Healthy	aws_nucleus_classic	4 hours ago

# Step 1: Configure Cloud Discovery

## 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.

i Configuring cloud discovery for client devices is not available for Greengrass core devices with Nucleus Lite runtime. Currently, it's available for devices with Nucleus Classic runtime. [Learn more](#)

#### 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 core device or a thing group.

# Step 2: 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 (0)** Info [Disassociate](#) [Associate client devices](#) [C](#)

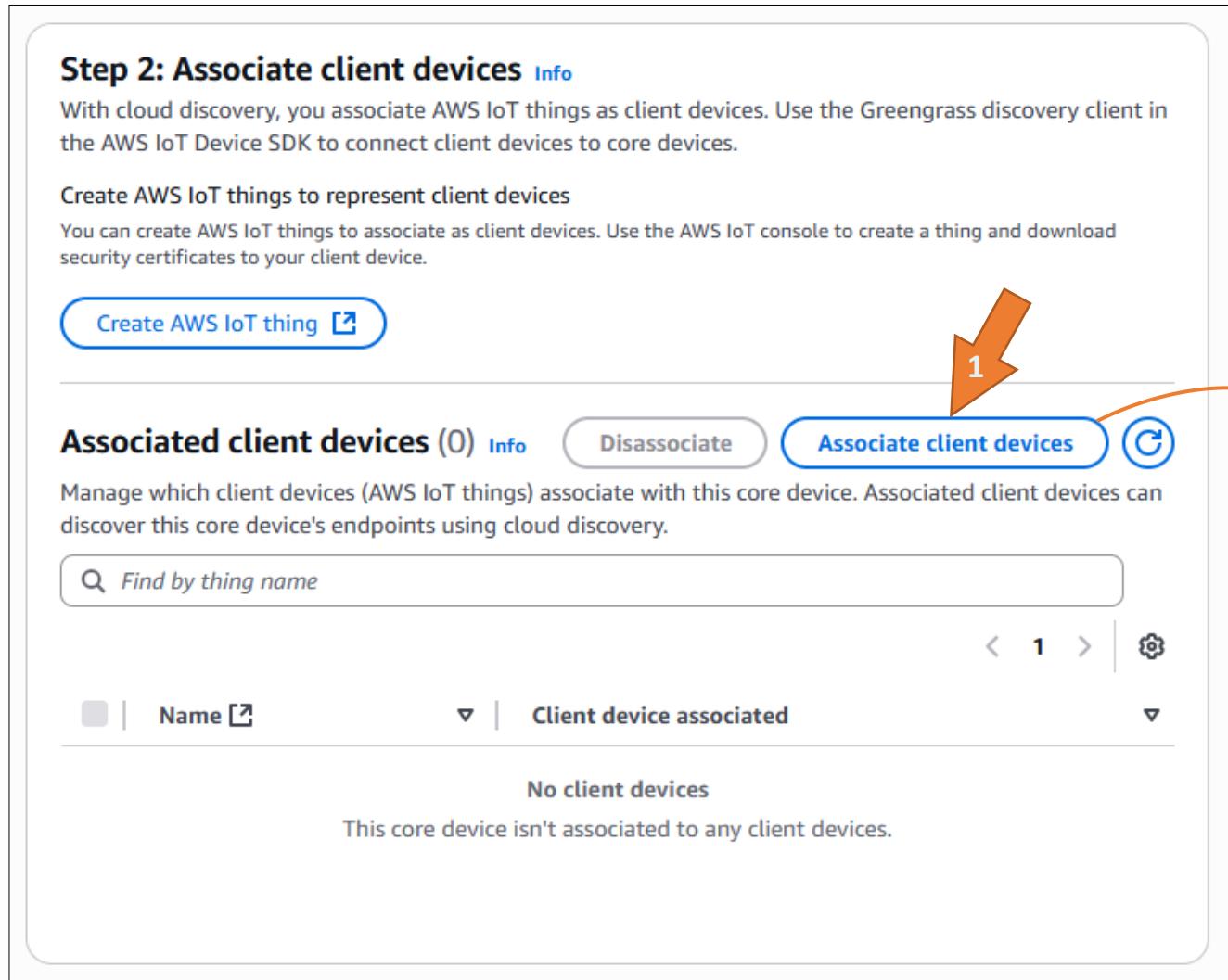
Manage which client devices (AWS IoT things) associate with this core device. Associated client devices can discover this core device's endpoints using cloud discovery.

Find by thing name

< 1 > | [⚙️](#)

Name [A](#) ▾ | Client device associated ▾

No client devices  
This core device isn't associated to any client devices.



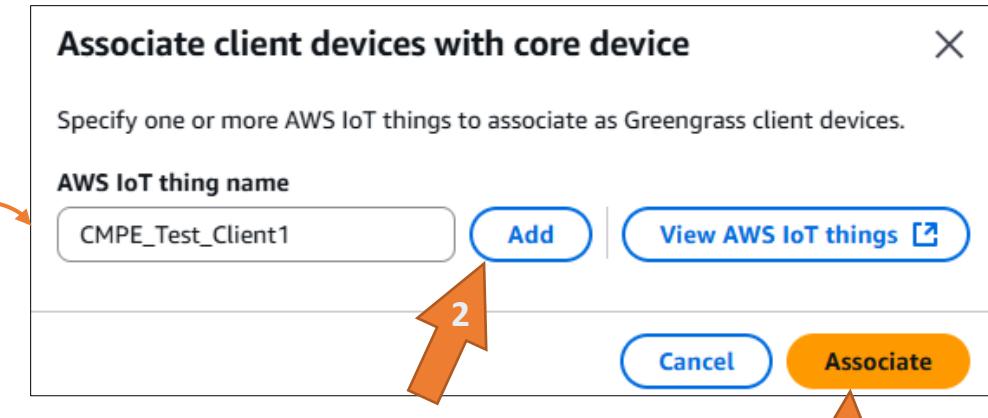
**Associate client devices with core device** X

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](#)

[Cancel](#) [Associate](#)



# Step 3: Deploy Required Greengrass Components

- The following Greengrass components are required:
  - Greengrass nucleus (aws.greengrass.Nucleus)
  - Client device auth (aws.greengrass.clientdevices.Auth)
  - IP detector (aws.greengrass.clientdevices.IPDetector)
- You must also deploy ONE of the following local brokers:
  - 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)
  - Shadow manager (aws.greengrass.ShadowManager)

# Step 3: Deploy Required Greengrass Components

## 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 3: Deploy Required Greengrass Components

## 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](#)

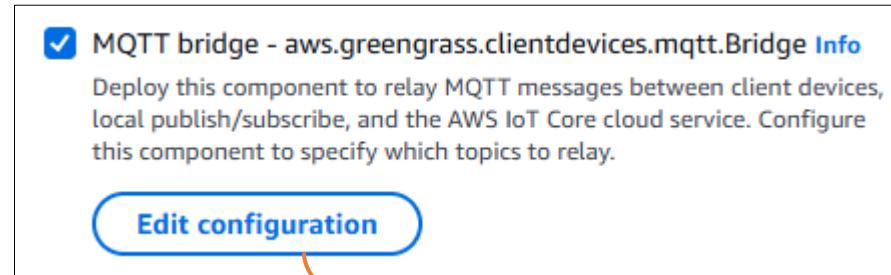
```
1 ▼ {  
2 ▼   "deviceGroups": {  
3     "formatVersion": "2021-03-05",  
4     "definitions": {  
5       "MyDeviceGroup": {  
6         "selectionRule": "thingName: CMPE_Test_Client*",  
7         "policyName": "MyClientDevicePolicy"  
8       }  
9     },  
10    "policies": {  
11      "MyClientDevicePolicy": {  
12        "AllowConnect": {  
13          "statementDescription": "Allow client devices to connect.",  
14          "operations": [  
15            "mqtt:connect"  
16          ],  
17          "resources": [  
18            "*"  
19          ]  
20        },  
21        "AllowPublish": {  
22          "statementDescription": "Allow client devices to publish to all"  
23        }  
24      }  
25    }  
26  }  
27 }  
28 }
```

JSON Ln 26, Col 12 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 3: Deploy Required Greengrass Components

## Customize aws.greengrass.clientdevices.mqtt.Bridge



- 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.

A screenshot of a JSON configuration editor titled 'Configuration to merge'. The text states: '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
```

The JSON code defines a single topic mapping named 'ClientDeviceHelloWorld' under 'mqttTopicMapping'. It maps the topic 'clients/+/hello/world' from the source 'LocalMqtt' to the target 'IotCore'. The code is numbered from 1 to 10. At the bottom, there are tabs for 'JSON', 'Ln 10, Col 1', 'Errors: 0', 'Warnings: 0', and a gear icon.

# Step 3: Deploy Required Greengrass Components

## 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](#)

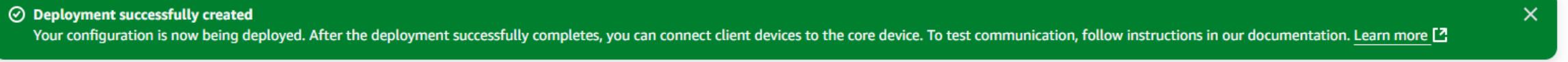


[Cancel](#) [Review and deploy](#)

# Step 3: Deploy Required Greengrass Components

## Check Deployment Status

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

 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](#)

**Deployment for CMPE583-GreenGrassCore**

Latest revision: 1 | Cancel | Actions

**Overview**

Target <a href="#">CMPE583-GreenGrassCore</a>	Target type Core device	Deployment created 2 minutes ago
Device status  Healthy	Deployment status  Completed	

**Components (5)**

Find by component name

Name	Version
<a href="#">aws.greengrass.Nucleus</a>	2.15.0
<a href="#">aws.greengrass.clientdevices.Auth</a>	2.5.4
<a href="#">aws.greengrass.clientdevices.IPDetector</a>	2.2.2
<a href="#">aws.greengrass.clientdevices.mqtt.Bridge</a>	2.3.2
<a href="#">aws.greengrass.clientdevices.mqtt.Moquette</a>	2.3.7

# Onboarding 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/greengrass  
  
(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 \  
--max_pub_ops 1
```

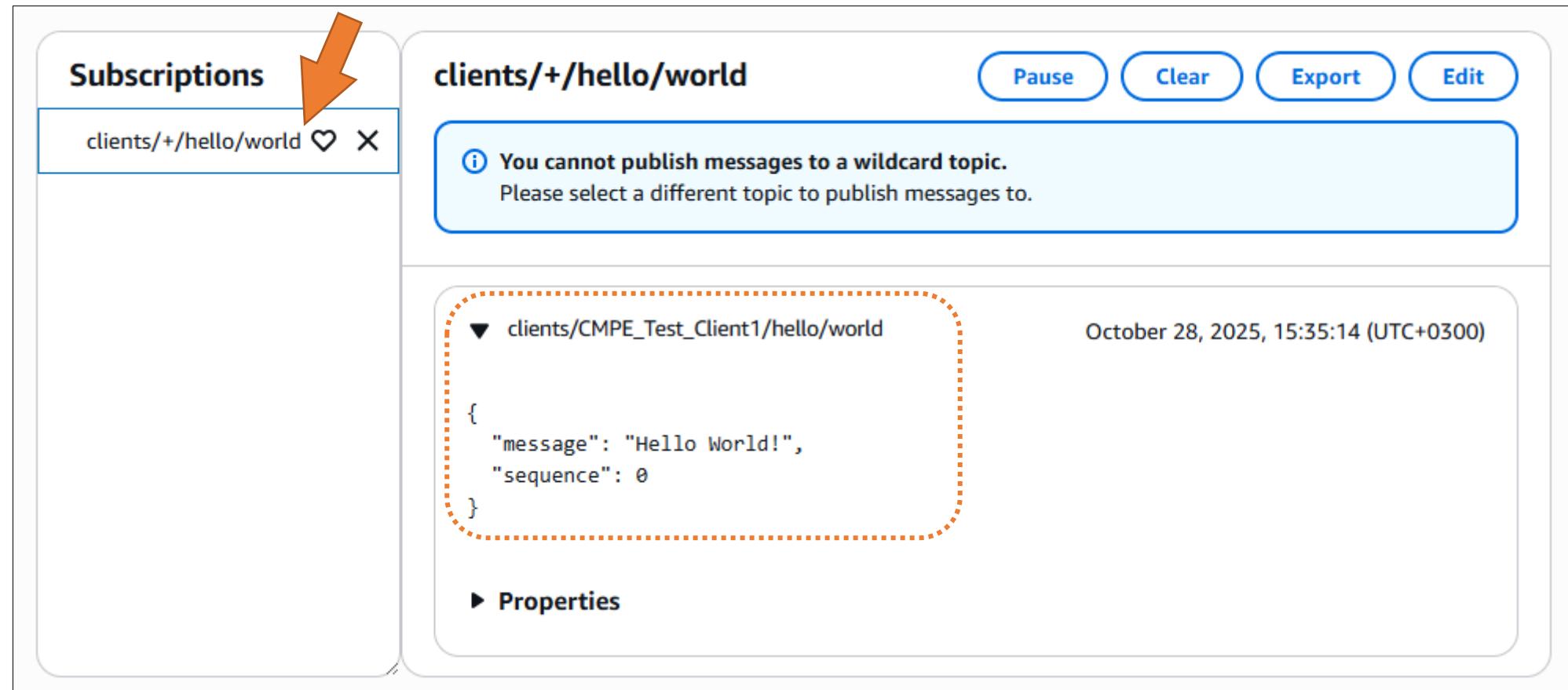
# Step 2: Run Sample Device Discovery

## Verify Client Device Connection From Terminal

```
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$ 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/greengrass/AmazonRootCA1.pem \
--cert /home/cagatay/shared_folder/greengrass/certificate1.pem.crt \
--key /home/cagatay/shared_folder/greengrass/private1.pem.key \
--region eu-central-1 \
--max_pub_ops 1
Performing greengrass discovery...
Received a greengrass discovery result! Not showing result for possible data sensitivity.
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/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Successfully published to topic clients/CMPE_Test_Client1/hello/world with payload `{"message": "Hello World!", "sequence": 0}`
```

# Step 3: Verify Client Device Connection (MQTT Bridge)

## Verify From MQTT Test Client



The screenshot shows a MQTT test client interface. On the left, there's a sidebar titled "Subscriptions" with a list containing "clients+/hello/world". An orange arrow points to this list. To the right, the main panel has a header "clients+/hello/world" with buttons for "Pause", "Clear", "Export", and "Edit". A message box displays an error: "You cannot publish messages to a wildcard topic. Please select a different topic to publish messages to." Below this, a message from "clients/CMPE\_Test\_Client1/hello/world" is shown, timestamped "October 28, 2025, 15:35:14 (UTC+0300)". The message content is: { "message": "Hello World!", "sequence": 0 }. A "Properties" button is also visible.

Subscriptions

clients+/hello/world

Pause Clear Export Edit

i You cannot publish messages to a wildcard topic.  
Please select a different topic to publish messages to.

clients/CMPE\_Test\_Client1/hello/world

{  
"message": "Hello World!",  
"sequence": 0  
}

Properties

October 28, 2025, 15:35:14 (UTC+0300)

# Communicate with Client Devices

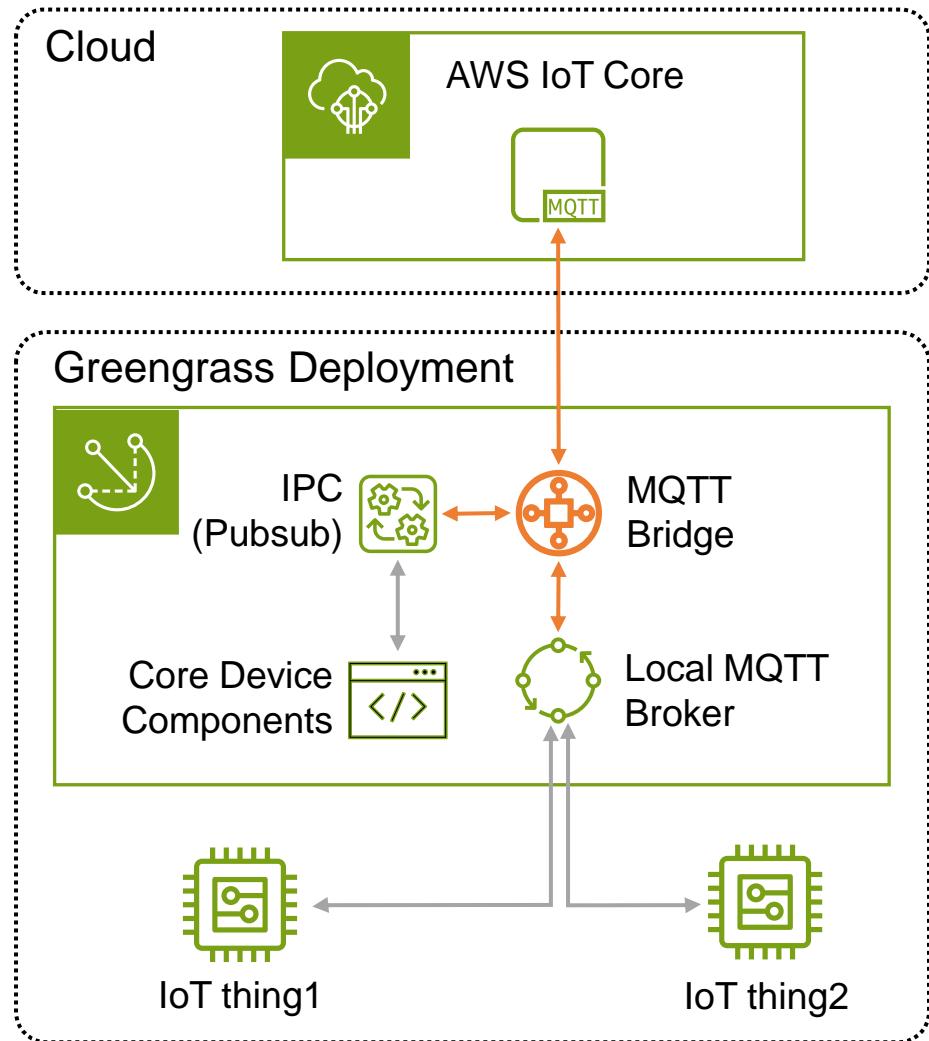
---

# MQTT Bridge: Understanding the Brokers

➤ The *MQTT bridge* component relays messages between three distinct systems. Before configuring the bridge, it's important to understand these systems:

1. Local MQTT ("**LocalMqtt**")
  - The **local** broker (e.g., Moquette) that client devices (like IoT sensors) connect to.
  - Handles messages between client devices and the core device.
2. Local Publish/Subscribe ("**Pubsub**")
  - The **internal** Greengrass message broker.
  - Handles messages between different components running inside the core device.
3. AWS IoT Core ("**IotCore**")
  - The **cloud-based** MQTT broker.
  - Handles messages between the Greengrass core device and the AWS Cloud.

# MQTT Bridge: Topic Mapping for Different Cases



Communication Scenario	<i>mqttTopicMapping</i> Settings of MQTT Bridge
Thing to Thing	Not required*
Component to Component	Not required**
Thing to Core Device	"source": "LocalMqtt" "target": "Pubsub"
Core Device to Thing	"source": "Pubsub" "target": "LocalMqtt"
Thing to IoT Core	"source": "LocalMqtt" "target": "IoTCore"
IoT Core to Thing	"source": "IoTCore" "target": "LocalMqtt"

\* Things can exchange messages directly via the Local MQTT broker; the MQTT Bridge is not required.

\*\* Components can exchange messages directly via the Local IPC (Pubsub).

# Steps for Client Device Communication

1

Create multiple IoT client devices

2

Configure cloud discovery to set up an MQTT bridge for topic mapping

3

Create custom component (with Python) to subscribe to local MQTT events

4

Deploy and review your component

5

Use the AWS IoT Device SDK to implement different use cases

# Step 1: Create More IoT Clients

Create Another Client and Associate It With Core Device

The screenshot shows the AWS IoT 'Create single thing' wizard. The top navigation bar includes 'AWS IoT > Manage > Things > Create things > Create single thing'. The left sidebar lists 'Step 1 Specify thing properties' (selected), 'Step 2 - optional Configure device certificate', and 'Step 3 - optional Attach policies to certificate'. The main content area is titled 'Specify thing properties' with a sub-section 'Thing properties'. A 'Thing name' input field contains 'CMPE\_Test\_Client2', with a note below stating: 'Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.' An orange arrow labeled '1' points to the 'Specify thing properties' title. Another orange arrow labeled '2' points to the 'Thing name' input field. A large orange curved arrow originates from the 'Associate' button in the second step and points back to the 'Thing name' input field.

Step 1  
Specify thing properties

Step 2 - optional  
Configure device certificate

Step 3 - optional  
Attach policies to certificate

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\_Client2

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

Associate client devices with core device

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

AWS IoT thing name

Enter the exact name of an AWS IoT thing

Add View AWS IoT things

CMPE\_Test\_Client2 X

Cancel Associate

# Step 2: Reconfigure Cloud Discovery

The screenshot shows the AWS IoT Greengrass Core devices configuration page. The left sidebar lists navigation options: Domain configurations, Test (MQTT test client, Device Location, Query connectivity status), Manage (All devices: Things, Thing groups, Thing types, Fleet metrics), and Greengrass devices (Core devices, Components, Deployments, Groups (V1)). A large orange arrow labeled '1' points to the 'Core devices' link in the Greengrass devices section.

The main content area displays the details for the device 'CMPE583-GreenGrassCore'. The title 'CMPE583-GreenGrassCore' is highlighted with a large orange arrow labeled '2'. The 'Overview' section contains a brief description of Greengrass core devices, the thing name 'CMPE583-GreenGrassCore', platform 'linux/amd64', runtime software 'aws\_nucleus\_classic 2.15.0', and logs (View in CloudWatch). The status is listed as 'Status' (Healthy) and 'Status reported' (37 minutes ago). Below the overview, there are tabs for Components, Deployments, Thing groups, Client devices (which is underlined in blue), and Tags. An orange arrow labeled '3' points to the 'Client devices' tab.

The 'Local authentication for client devices' section explains how client devices can authenticate with a Greengrass core device without connecting to the AWS IoT Greengrass cloud service. It includes a 'View documentation' button. An orange arrow labeled '4' points to this button.

The 'Cloud discovery configuration' section describes how core devices store connectivity information in the AWS IoT Greengrass cloud service for client devices to discover connectivity information. It includes an 'Info' link and a 'Configure cloud discovery' button. An orange arrow labeled '4' also points to this button.

# Step 2: Reconfigure Cloud Discovery

## Reconfigure 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](#)

**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](#)

```
7      },
8      "ClientDeviceEvents": {
9          "topic": "clients/+/alert",
10         "targetTopicPrefix": "event/",
11         "source": "LocalMqtt",
12         "target": "Pubsub"
13     },
14     "ClientDeviceCloudStatusUpdate": {
15         "topic": "clients/+/status",
16         "targetTopicPrefix": "update/",
17         "source": "LocalMqtt",
18         "target": "IoTCORE"
19     }
20 }
21 }
22 }
```

JSON Ln 2, Col 14 0 Errors: 0 0 Warnings: 0

# Step 2: 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](#)

```
7      },
8  "ClientDeviceEvents": {
9    "topic": "clients/+/alert",
10   "targetTopicPrefix": "event/",
11   "source": "LocalMqtt",
12   "target": "Pubsub"
13  },
14  "ClientDeviceCloudStatusUpdate": {
15    "topic": "clients/+/status",
16    "targetTopicPrefix": "update/",
17    "source": "LocalMqtt",
18    "target": "IotCore"
19  }
20}
21}
22}
```

JSON Ln 2, Col 14 Errors: 0 | Warnings: 0

**Relay messages from client devices to local IPC (Pubsub) through MQTT Bridge with **clients/+/alert** topic filter and add the *events/* prefix to the target topic. Core device components can receive this message with **events/clients/+/alert** topic.**

**Relay messages from client devices to AWS IoT Core through MQTT Bridge with **clients/+/status** topic filter and add *update/* prefix to the target topic. IoT Core can receive this message with **update/clients/+/status** topic.**

# Step 2: 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 2: Reconfigure Cloud Discovery

## Check Deployment Status

### Deployment for CMPE583-GreenGrassCore

Latest revision: 2 ▾ Cancel Actions ▾

**Overview**

<b>Target</b> CMPE583-GreenGrassCore	<b>Target type</b> Core device	<b>Deployment created</b> in 0 seconds
<b>Device status</b> Healthy	<b>Deployment status</b> Completed	

**Components (5)**

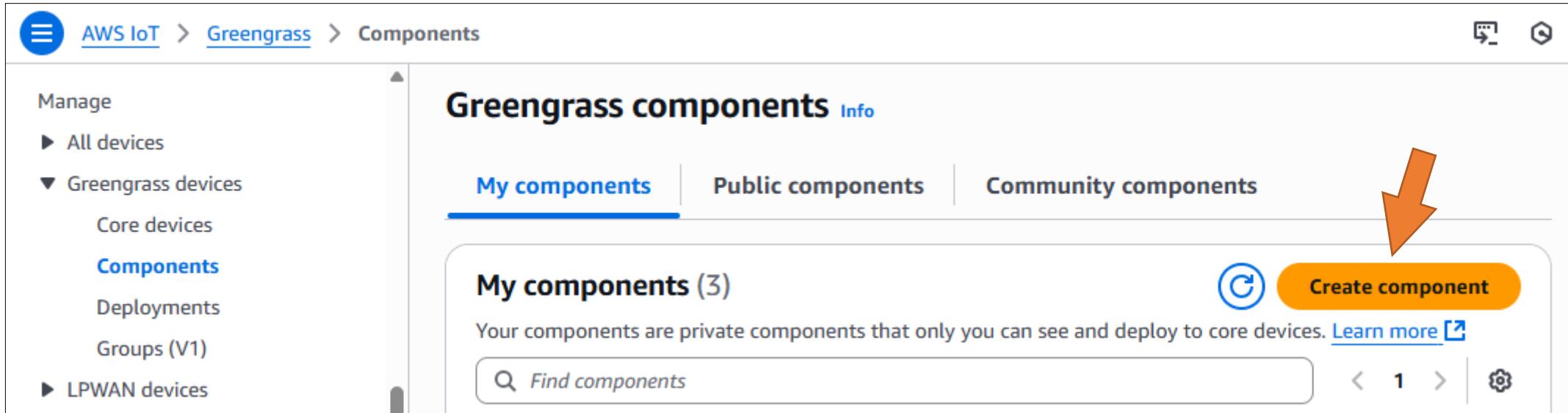
Find by component name

Name	Version
<a href="#">aws.greengrass.Nucleus</a>	2.15.0
<a href="#">aws.greengrass.clientdevices.Auth</a>	2.5.4
<a href="#">aws.greengrass.clientdevices.IPDetector</a>	2.2.2
<a href="#">aws.greengrass.clientdevices.mqtt.Bridge</a>	2.3.2
<a href="#">aws.greengrass.clientdevices.mqtt.Moquette</a>	2.3.7



# Step 3: Create a Custom Component

To Suscribe Local MQTT Events



The screenshot shows the AWS IoT Greengrass Components interface. The left sidebar has a 'Manage' section with 'All devices', 'Greengrass devices' (expanded to show 'Core devices', 'Components' which is highlighted in blue, 'Deployments', 'Groups (V1)', and 'LPWAN devices'). The main area is titled 'Greengrass components' with an 'Info' link. It has three tabs: 'My components' (selected), 'Public components', and 'Community components'. Below the tabs, it says 'My components (3)'. A message states: 'Your components are private components that only you can see and deploy to core devices.' with a 'Learn more' link. There is a search bar with 'Find components' placeholder text. On the right, there is a large orange 'Create component' button with a circular icon containing a 'C'. An orange arrow points from the bottom right towards this button.

# Step 3: Create a Custom Component

## Configure accessControl for Local Messages

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

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 3: Create a Custom Component

## Configure Artifacts in the Recipe

**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](#)

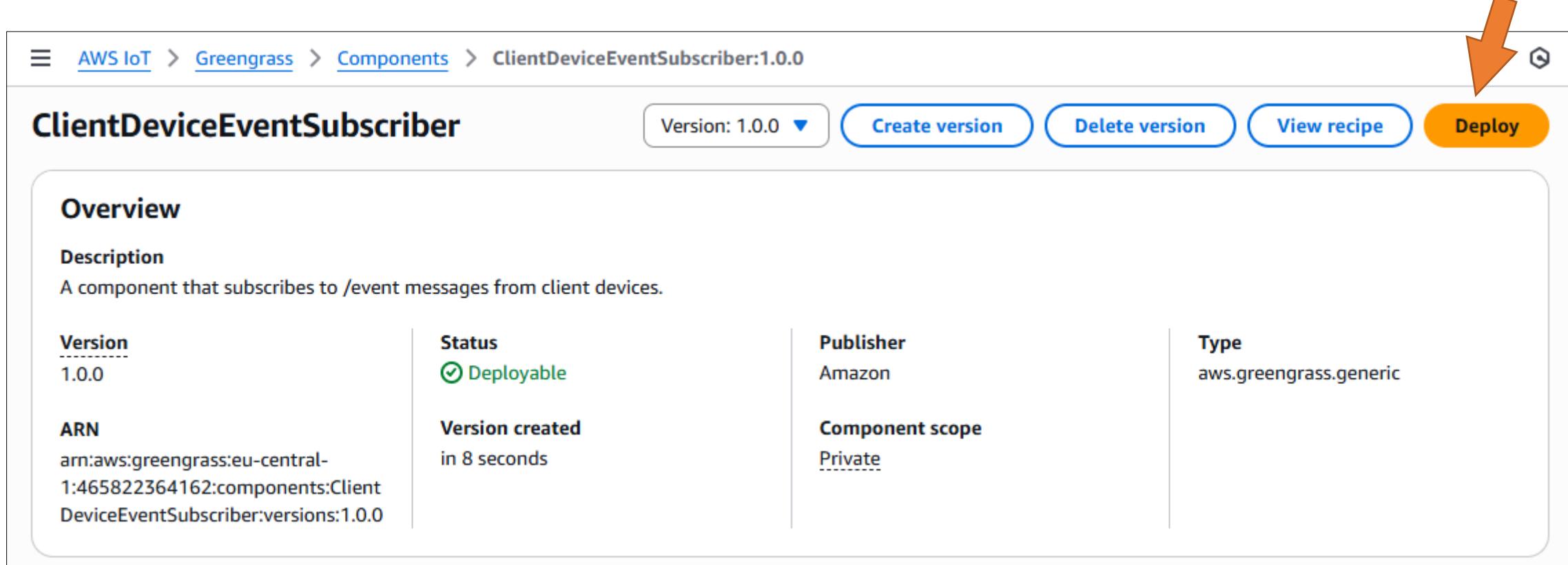
```
30 ▼   "Manifests": [  
31 ▼     {  
32 ▼       "platform": {  
33         "os": "linux"  
34       },  
35 ▼       "Lifecycle": {  
36         "Install": "pip3 install --user awsiotsdk --break-system-packages",  
37         "Run": "python3 -u {artifacts:path}/cmpe583-ClientDeviceEventSubscriber.py"  
38       },  
39 ▼       "Artifacts": [  
40 ▼         {  
41           "Uri": "s3://cmpe583.greengrass.bucket/artifacts/cmpe583-  
42             -ClientDeviceEventSubscriber.py",  
43             "Digest": "0nsJZs7zXTg/QiSJ7FWX7gneJFCSSLmTGJKZZrsF5k8=",  
44             "Algorithm": "SHA-256",  
45             "Unarchive": "NONE",  
46             "Permission": {  
47               "Read": "OWNER",  
48               "Execute": "NONE"  
49             }  
50           }  
51         ]  
52     }  
53   ]
```

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 4: Deploy Your Component

## Via AWS IoT Greengrass (Console)



The screenshot shows the AWS IoT Greengrass Components page for the 'ClientDeviceEventSubscriber' component version 1.0.0. The page includes a navigation bar, a header with the component name, and a main content area with tabs for Overview, Description, Version, Status, Publisher, and Type. A large orange arrow points to the 'Deploy' button.

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.

<b>Version</b> 1.0.0	<b>Status</b>  Deployable	<b>Publisher</b> Amazon	<b>Type</b> aws.greengrass.generic
<b>ARN</b> arn:aws:greengrass:eu-central-1:465822364162:components:ClientDeviceEventSubscriber:versions:1.0.0	<b>Version created</b> in 8 seconds	<b>Component scope</b> Private	

# Step 4: Deploy Your Component

## Select the Component

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 (4) Info**

Name	Supported runtime	Operating system	Architecture
<a href="#">ClientDeviceEventSubscriber</a>	aws_nucleus_classic	linux	All

**Public components (48) Info**

Name	Supported runtime	Operating system	Architecture
<a href="#">aws.greengrass.Nucleus</a>	aws_nucleus_classic	linux, darwin, windows	All
<a href="#">aws.greengrass.clientdevices.Auth</a>	aws_nucleus_classic	All	All
<a href="#">aws.greengrass.clientdevices.IPDetector</a>	aws_nucleus_classic	All	All
<a href="#">aws.greengrass.clientdevices.mqtt.Bridge</a>	aws_nucleus_classic	All	All
<a href="#">aws.greengrass.clientdevices.mqtt.Moquette</a>	aws_nucleus_classic	All	All

Show only selected components < 1 >



# Step 4: Deploy Your Component

## Review & Deploy

### Step 4: Advanced deployment configurations

[Edit](#)

**Advanced deployment configurations**

**Component update policy**  
Notify components, Component update response timeout: 60 seconds

**Configuration validation response timeout**  
30 seconds

**Failure handling policy**  
Rollback

[Download as JSON](#)

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



# Step 4: Deploy Your Component

## Check Deployment Status on AWS IoT Console

The screenshot shows the AWS IoT Greengrass Deployment status page for a deployment named "Deployment for CMPE583-GreenGrassCore".

**Deployment Overview:**

- Target:** CMPE583-GreenGrassCore
- Target type:** Core device
- Deployment created:** in 5 seconds
- Device status:** Healthy
- Deployment status:** Completed

**Components (6):**

- ClientDeviceEventSubscriber
- aws.greengrass.Nucleus
- aws.greengrass.clientdevices.Auth

**Deployment Revision:** Latest revision: 3

**Actions:** Cancel, Actions

**Table Headers:** Name, Version

Name	Version
ClientDeviceEventSubscriber	1.0.0
aws.greengrass.Nucleus	2.15.0
aws.greengrass.clientdevices.Auth	2.5.4

# Step 4: 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.

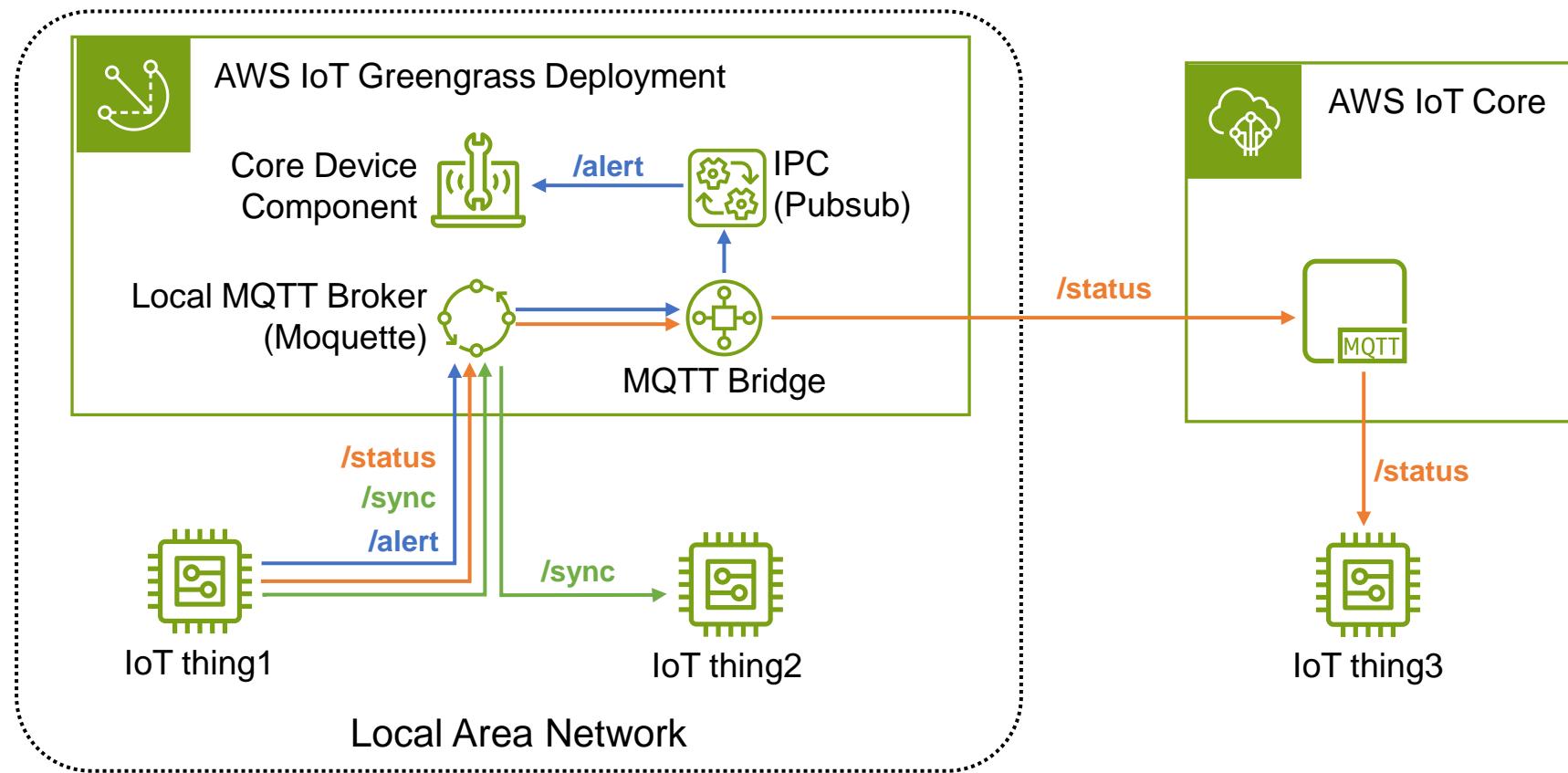
```
root@cagatay:/home/cagatay# tail -n 100 /greengrass/v2/logs/ClientDeviceEventSubscriber.log
2025-10-28T13:23:57.755Z [INFO] (pool-3-thread-79) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW, command=["pip3 install --user awsiotsdk --break-system-packages"]}
2025-10-28T13:23:59.243Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: awsiotsdk in /home/ggc_user/.local/lib/python3.12/site-packages (1.26.0). {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2025-10-28T13:23:59.243Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: awscli==0.28.1 in /home/ggc_user/.local/lib/python3.12/site-packages (from awsiotsdk) (0.28.1). {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2025-10-28T13:23:59.743Z [INFO] (pool-3-thread-79) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=STARTING, command=["python3 -u /greengrass/v2/packages/artifacts/ClientDeviceEventSubscriber/1.0.0..."]}
2025-10-28T13:24:00.110Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Successfully subscribed to topic: event/clients/+/alert. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=RUNNING}
root@cagatay:/home/cagatay#
```

# Four Different Device Communication Scenarios

---

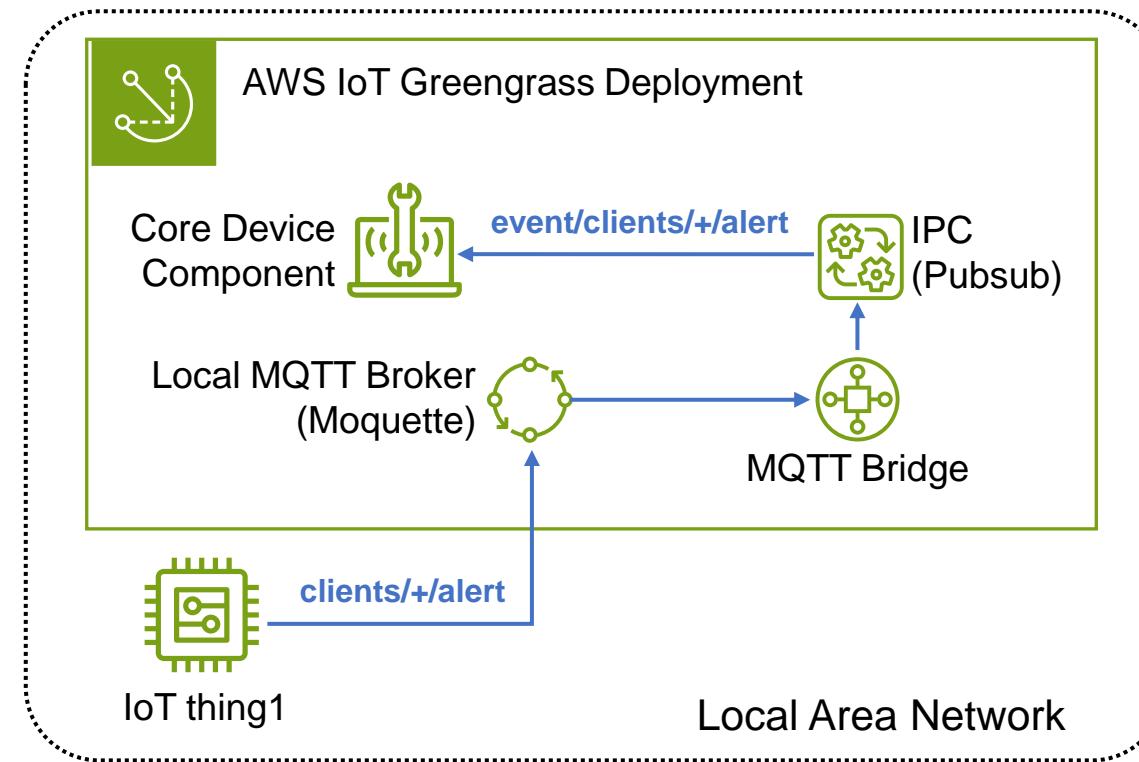
# Communicating with Client Devices

- In this section, we will examine 4 different communication scenario where we will send events from the IoT device to the core device and the cloud using the **MQTT Bridge**, and to other local client devices directly via the **Local MQTT Broker**.



# Case 1: IoT Client to Core Device Communication

- Publish 'clients/+/alert' Topic From IoT Client1
- Receive '**event**/clients/+/alert' on Core Device Component



# Step 1: 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 \
--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 1: Publish 'clients/+/alert' Topic From IoT Client

## Verify Process on Terminal

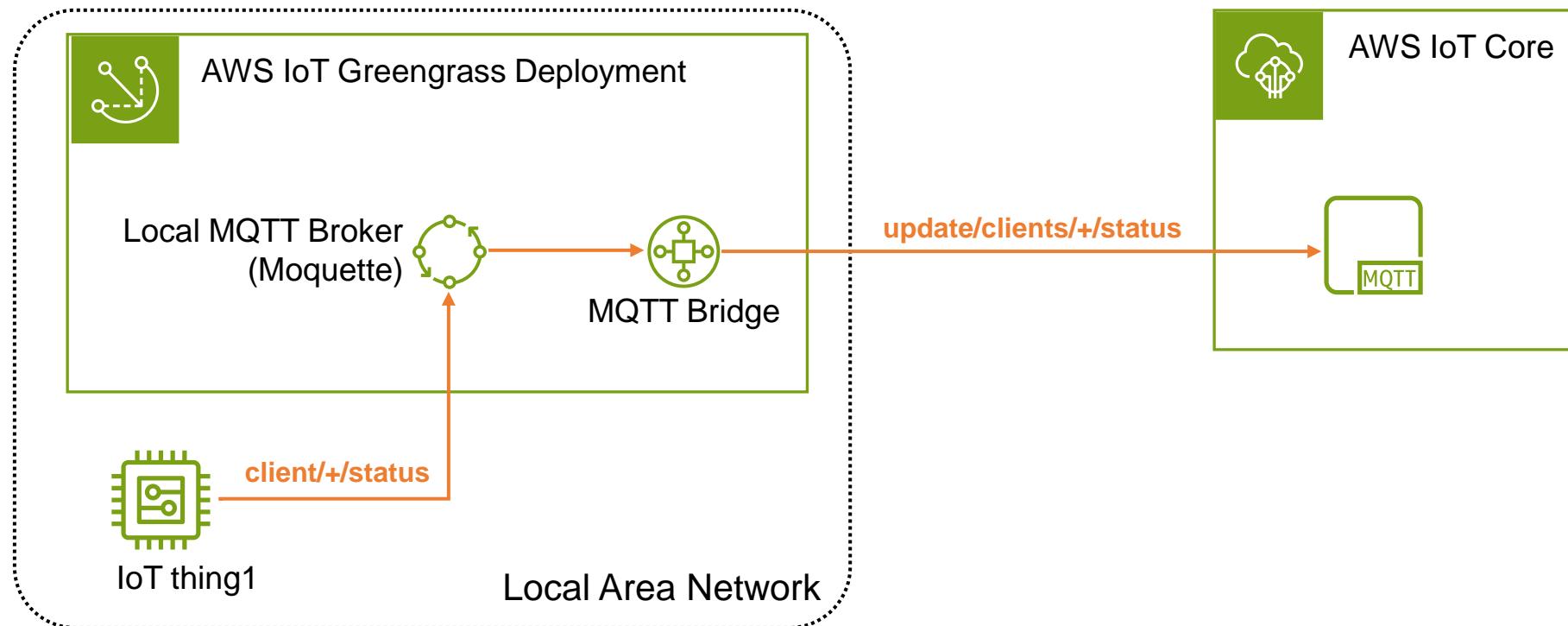
```
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$ python3 basic_discovery.py \
--thing_name CMPE_Test_Client1 \
--topic 'clients/CMPE_Test_Client1/alert' \
--message 'Production Anomaly Detected. Check the Production Plant!' \
--ca_file /home/cagatay/shared_folder/greengrass/AmazonRootCA1.pem \
--cert /home/cagatay/shared_folder/greengrass/certificate1.pem.crt \
--key /home/cagatay/shared_folder/greengrass/private1.pem.key \
--region eu-central-1 \
--max_pub_ops 1 \
--mode publish
Performing greengrass discovery...
Received a greengrass discovery result! Not showing result for possible data sensitivity.
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreenGrassCore at host 10.0.2.15 port 8883
Connected!
Successfully published to topic clients/CMPE_Test_Client1/alert with payload `{"message": "Production Anomaly Detected. Check the Production Plant!", "sequence": 0}`
```

# Step 2: Receive 'event/clients/+/alert' on Component Verify Process on Core Device's Terminal

```
root@cagatay:/home/cagatay#
root@cagatay:/home/cagatay# tail -n 100 /greengrass/v2/logs/ClientDeviceEventSubscriber.log
2025-10-28T13:23:57.755Z [INFO] (pool-3-thread-79) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW, command=["pip 3 install --user awsiotsdk --break-system-packages"]}
2025-10-28T13:23:59.243Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: awsiotsdk in /home/ggc_user/.local/lib/python3.12/site-packages (1.26.0). {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2025-10-28T13:23:59.243Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: awscrt==0.28.1 in /home/ggc_user/.local/lib/python3.12/site-packages (from awsiotsdk) (0.28.1). {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2025-10-28T13:23:59.743Z [INFO] (pool-3-thread-79) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=STARTING, command=["python3 -u /greengrass/v2/packages/artifacts/ClientDeviceEventSubscriber/1.0.0..."]}
2025-10-28T13:24:00.110Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Successfully subscribed to topic: event/clients/+/alert. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=RUNNING}
2025-10-28T13:58:19.828Z [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}
root@cagatay:/home/cagatay#
```

# Case 2: IoT Client to IoT Core Communication

- Publish 'clients/+status' Topic From IoT Client1
- Receive '**update/clients/+status**' Topic on AWS IoT Core



# Step 1: 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 \
    --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 1: Publish 'clients+/status' Topic From IoT Client

## Verify Process on Terminal

```
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$ 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/greengrass/AmazonRootCA1.pem \
--cert /home/cagatay/shared_folder/greengrass/certificate1.pem.crt \
--key /home/cagatay/shared_folder/greengrass/private1.pem.key \
--region eu-central-1 \
--max_pub_ops 1 \
--mode publish
Performing greengrass discovery...
Received a greengrass discovery result! Not showing result for possible data sensitivity.
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreenGrassCore at host 10.0.2.15 port 8883
Connected!
Successfully published to topic clients/CMPE_Test_Client1/status with payload `{"message": "Client device is running for 2 days without error!"`  
, "sequence": 0}`
```

# Step 2: Receive 'update/clients/+status' on IoT Core

## Verify Process on MQTT Test Client

The screenshot shows the MQTT Test Client interface. On the left, under 'Subscriptions', there is a single entry: 'update/clients/+status'. An orange arrow points to this entry. To the right, under the topic 'update/clients/+status', there is a message box containing the following text:

**i You cannot publish messages to a wildcard topic.**  
Please select a different topic to publish messages to.

Below this message, a message is listed for the topic 'update/clients/CMPE\_Test\_Client1/status':

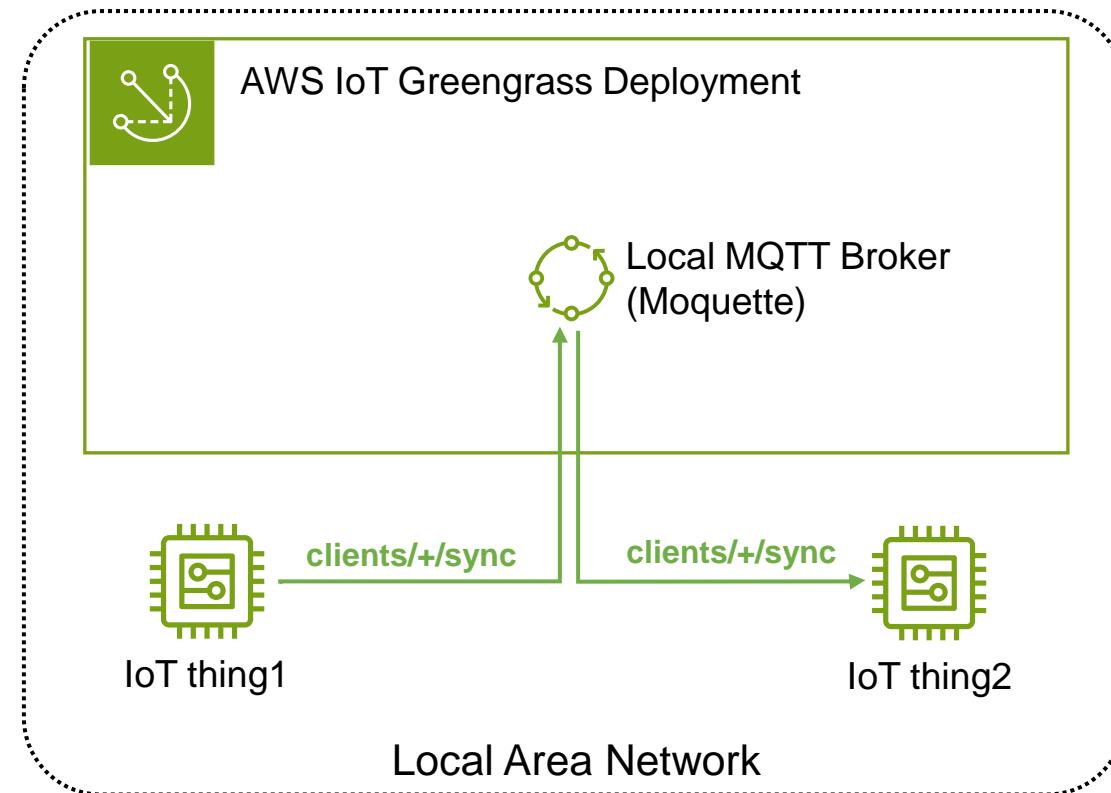
▼ update/clients/CMPE\_Test\_Client1/status      October 28, 2025, 17:04:41 (UTC+0300)

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

Below the message, there is a 'Properties' button.

# Case 3: IoT Client to IoT Client Communication

- Publish 'clients/+sync' Topic From IoT Client1 to IoT Client2



# Step 1: Publish 'clients/+/sync' Topic From IoT Client1

## 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 test client 2 through local Greengrass core MQTT broker

★ Things can exchange messages directly via the Local MQTT broker; the MQTT Bridge is not required.

# Step 1: Publish 'clients/+/sync' Topic From IoT Client1

## Verify Topic Published

```
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$ python3 basic_discovery.py \
    --thing_name 'CMPE_Test_Client1' \
    --topic 'clients/CMPE_Test_Client1/sync' \
    --message 'Hello from IoT Client 1!' \
    --ca_file /home/cagatay/shared_folder/greengrass/AmazonRootCA1.pem \
    --cert /home/cagatay/shared_folder/greengrass/certificate1.pem.crt \
    --key /home/cagatay/shared_folder/greengrass/private1.pem.key \
    --region eu-central-1 \
    --max_pub_ops 1 \
    --mode publish

Performing greengrass discovery...
Received a greengrass discovery result! Not showing result for possible data sensitivity.
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreenGrassCore at host 10.0.2.15 port 8883
Connected!
Successfully published to topic clients/CMPE_Test_Client1/sync with payload `{"message": "Hello from IoT Client 1!", "sequence": 0}`

(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$
```

# Step 2: Receive 'clients/+/sync' Topic on IoT Client2

## 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
```

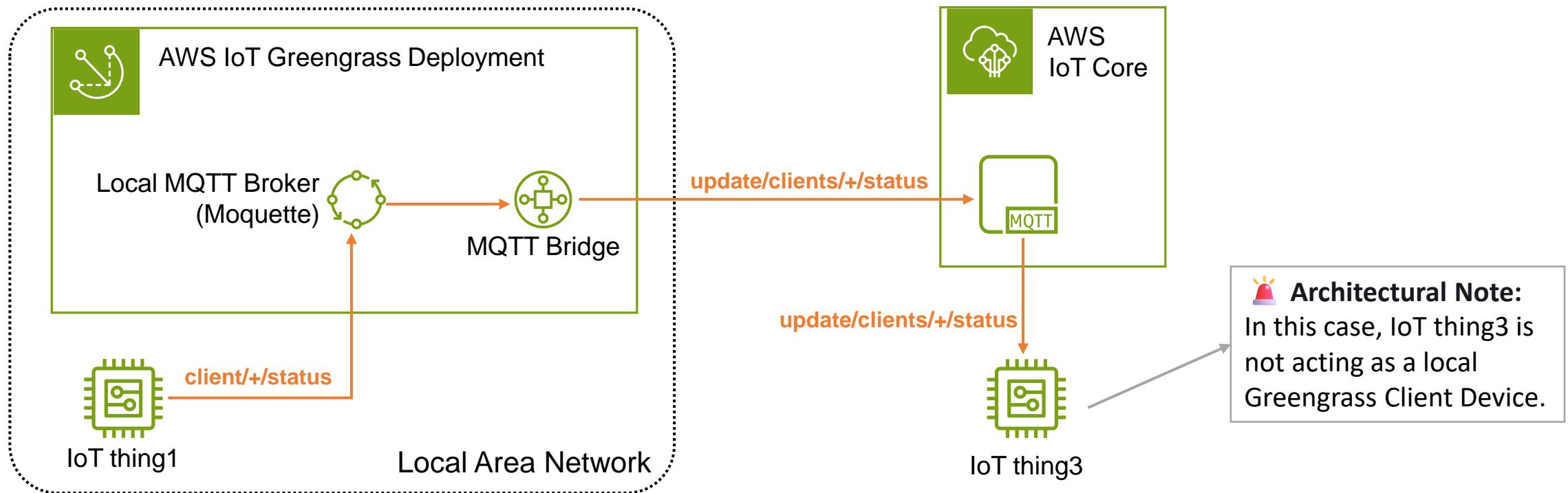
# Step 2: Receive 'clients/+/sync' Topic on IoT Client2

## Verify Topic Received

```
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$ python3 basic_discovery.py \
    --thing_name 'CMPE_Test_Client2' \
    --topic 'clients/+/sync' \
    --ca_file /home/cagatay/shared_folder/greengrass/AmazonRootCA1.pem \
    --cert /home/cagatay/shared_folder/greengrass/certificate2.pem.crt \
    --key /home/cagatay/shared_folder/greengrass/private2.pem.key \
    --region eu-central-1 \
    --mode subscribe
Performing greengrass discovery...
Received a greengrass discovery result! Not showing result for possible data sensitivity.
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}'
```

# Case 4: IoT Core to IoT Client Communication

- Publish 'clients/+status' Topic from IoT Client1 to IoT Core
- Receive 'update/clients/+status' topic on IoT Client3



# Step 1: Publish 'clients+/status' Topic From IoT Client1

## Publish Topic to Trigger Subscribers (AWS IoT Core -> IoT Client 3)

- 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 1: Publish 'clients+/status' Topic From IoT Client1

## Verify Topic Published

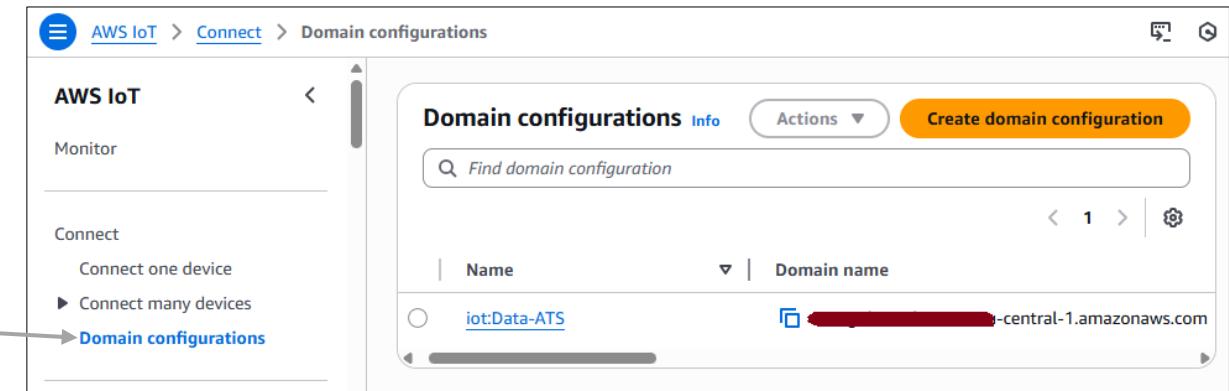
```
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$ python3 basic_discovery.py \
--thing_name CMPE_Test_Client1 \
--topic 'clients/CMPE_Test_Client1/status' \
--message 'Client device is running for 2 days without error!' \
--ca_file /home/cagatay/shared_folder/greengrass/AmazonRootCA1.pem \
--cert /home/cagatay/shared_folder/greengrass/certificate1.pem.crt \
--key /home/cagatay/shared_folder/greengrass/private1.pem.key \
--region eu-central-1 \
--max_pub_ops 1 \
--mode publish

Performing greengrass discovery...
Received a greengrass discovery result! Not showing result for possible data sensitivity.
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreenGrassCore at host 10.0.2.15 port 8883
Connected!
Successfully published to topic clients/CMPE_Test_Client1/status with payload `{"message": "Client device is running for 2 days without error!", "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 domain configuration settings

```
(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 2: Receive 'update/clients/+/status' on IoT Client3

## Subscribe Topic From Test Client 3

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

```
(venvclient3)$ cd aws-iot-device-sdk-python-v2/samples
```

```
(venvclient3)$ python3 pubsub.py \
    --topic 'update/clients/+/status' \
    --message '' \
    --endpoint 'your-endpoint' \
    --ca_file ~/AmazonRootCA1.pem \
    --cert ~/certificate.pem.crt \
    --key ~/private.pem.key
```

# Step 2: Receive 'update/clients/+/status' on IoT Client3

## Verify Topic Received

```
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples$ python3 pubsub.py \
    --topic 'update/clients/+/status' \
    --message '' \
    --endpoint '[REDACTED].iot.eu-central-1.amazonaws.com' \
    --ca_file /home/cagatay/shared_folder/greengrass/AmazonRootCA1.pem \
    --cert /home/cagatay/shared_folder/greengrass/certificate2.pem.crt \
    --key /home/cagatay/shared_folder/greengrass/private2.pem.key

Starting MQTT5 X509 PubSub Sample

===== Creating MQTT5 Client =====

===== Starting client =====
Lifecycle Connection Attempt
Connecting to endpoint: '[REDACTED] iot.eu-central-1.amazonaws.com' with client ID'mqtt5-sample-fa241d03'
Lifecycle Connection Success with reason code:<ConnectReasonCode.SUCCESS: 0>

===== Subscribing to topic 'update/clients/+/status' =====
Suback received with reason code:[<SubackReasonCode.GRANTED_QOS_1: 1>]

===== Sending 5 message(s) =====

===== Received message from topic 'update/clients/CMPE_Test_Client1/status': {"message": "Client device is running for 2 days without error!", "sequence": 0} =====
```

# Step 2: Receive 'update/clients/+status' on IoT Client3

## Troubleshooting

- The pubsub.py sample code by default both publishes and subscribes.
- Remove publishing part from the source code to focus only on receiving messages.

```
146     # Publish
147     if message_count == 0:
148         print("==== Sending messages until program killed ====\n")
149     else:
150         print("==== Sending {} message(s) ====".format(message_count))
151
152         publish_count = 1
153         while (publish_count <= message_count) or (message_count == 0):
154             message = f"{message_string} [{publish_count}]"
155             print(f"Publishing message to topic '{message_topic}': {message}")
156             publish_future = client.publish(mqtt5.PublishPacket(
157                 topic=message_topic,
158                 payload=message,
159                 qos=mqtt5.QoS.AT_LEAST_ONCE
160             ))
161             publish_completion_data = publish_future.result(TIMEOUT)
162             print("PubAck received with {}".format(repr(publish_completion_data.puback.reason_code)))
163             time.sleep(1.5)
164             publish_count += 1
165
166             received_all_event.wait(TIMEOUT)
167             print("{} message(s) received.\n".format(received_count))
168
```

# Thank You

