



# Edge Computing with AWS IoT Greengrass - Part #1

by Cagatay Sonmez, 7 Nov 2024

# The Primary Sources Used in This Lecture

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

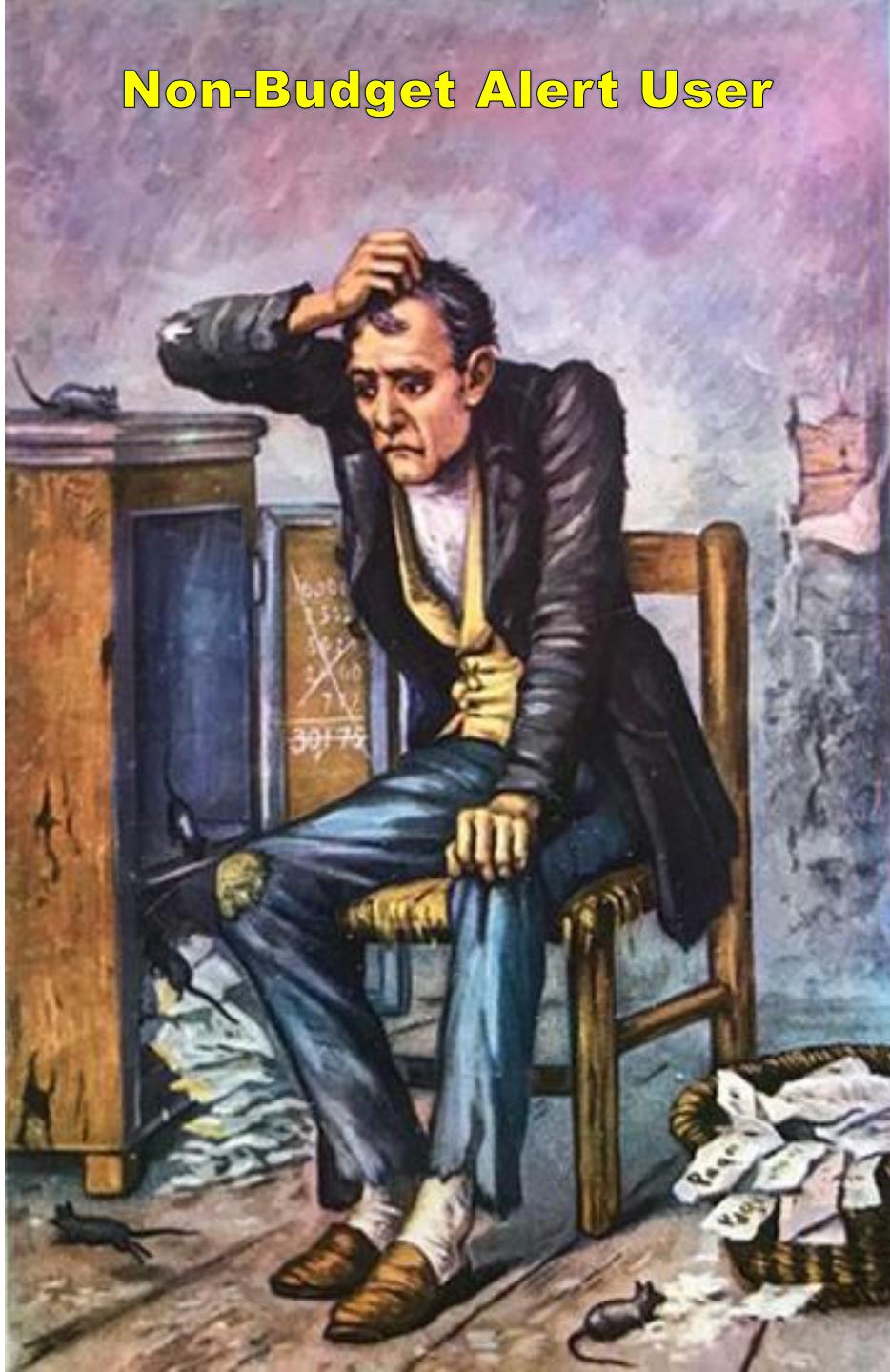
# ⚠ Reminder

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

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

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

**Non-Budget Alert User**

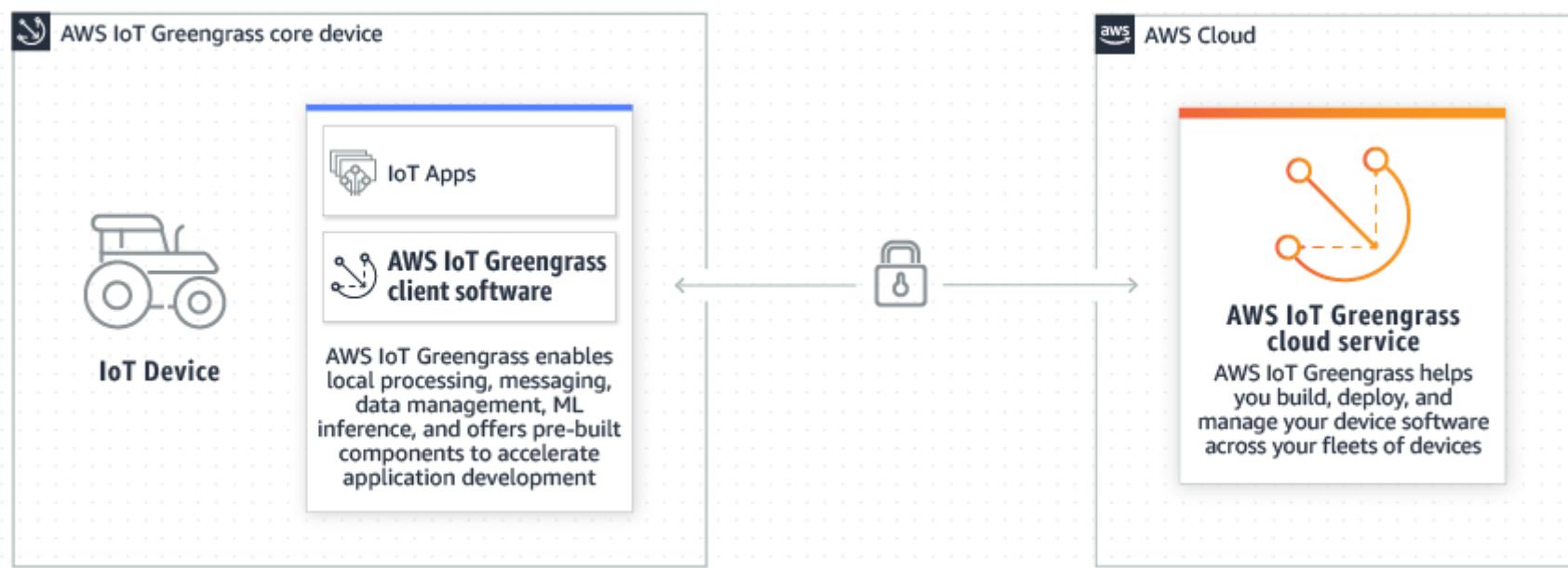


**Budget Alert User**



# What is AWS IoT Greengrass?

- AWS IoT Greengrass is software that extends cloud capabilities to local devices. This enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. Local devices can also communicate securely with AWS IoT Core and export IoT data to the AWS Cloud<sup>1</sup>.

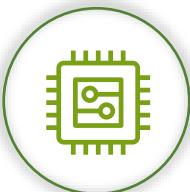


<sup>1</sup> <https://docs.aws.amazon.com/greengrass/v2/developerguide/what-is-iot-greengrass.html>

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



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

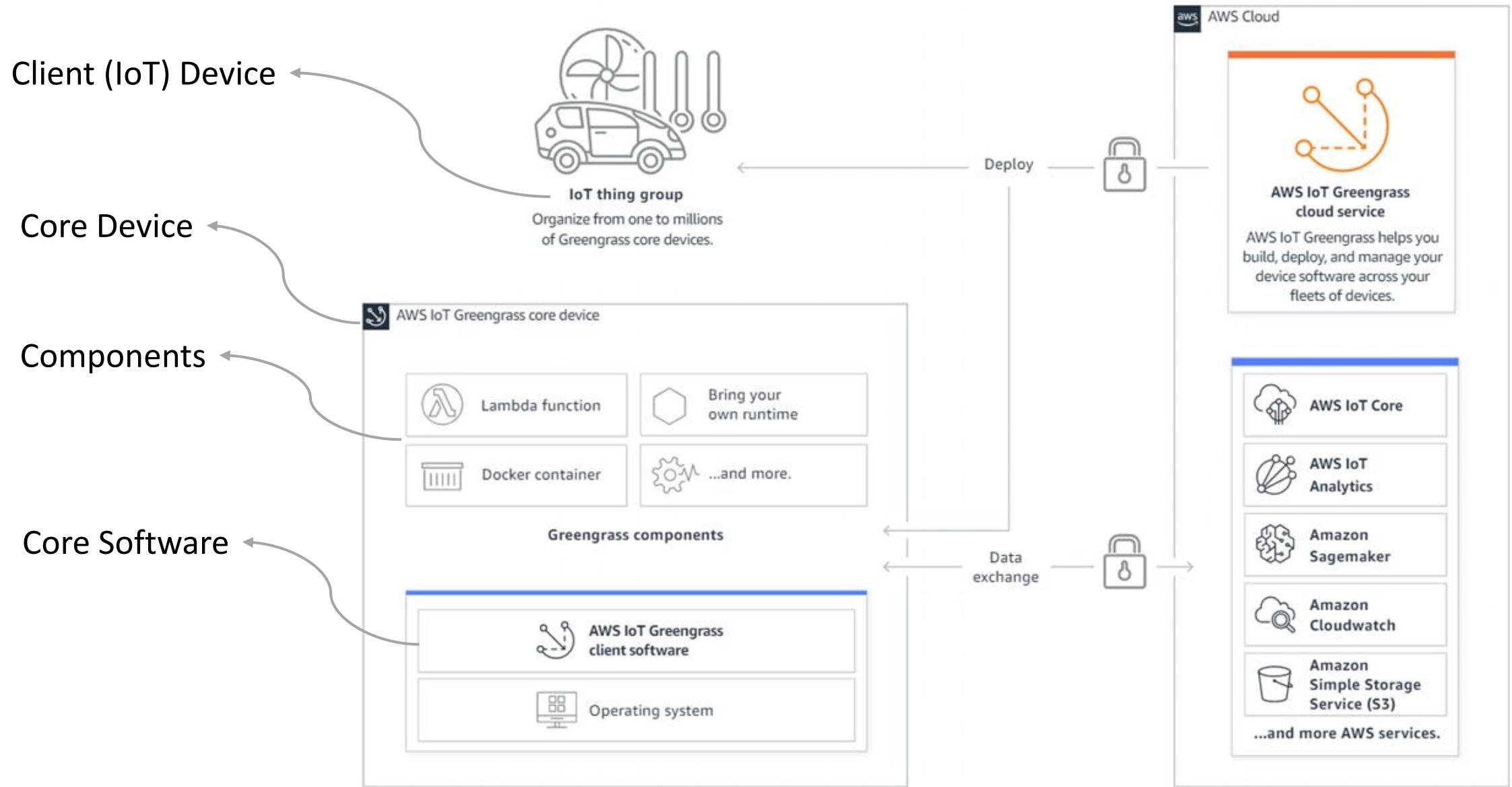


**Greengrass Component:** A software module (e.g. Python script) that is deployed to and runs on a Greengrass core device. These modules are modeled as components which can perform various tasks, such as collecting data from sensors, controlling actuators, or running machine learning models.



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

# How AWS IoT Greengrass works



# Selecting Client's Operating System

- Some features are supported on only certain operating systems!

Security		Linux	Windows
Feature	Use a hardware security module (HSM) to securely store the device's private key and certificate		
<b>Component features</b>			
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	
Publish data to Amazon Kinesis Data Firehose delivery streams using the Kinesis Data Firehose component	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	
Publish video streams to Amazon Kinesis Video Streams using the edge connector for Kinesis Video Streams component	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	
<i>Run AWS IoT Greengrass in a Docker container using a prebuilt Docker image</i>			
Feature	Linux	Windows	
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 IoT secure tunneling			<input checked="" type="checkbox"/> Yes
Feature	Linux	Windows	
Manage core devices with AWS Systems Manager	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	
Connect to core devices with AWS IoT secure tunneling	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	

Machine learning		Linux	Windows
Feature	Perform machine learning inference using Amazon Lookout for Vision		
<i>Perform machine learning inference using Amazon Lookout for Vision</i>			<input checked="" type="checkbox"/> Yes
Feature	Linux	Windows	
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...

# Step 1: Set Up Your Environment

## For Linux Based Devices

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

For Debian-based or Ubuntu-based distributions:

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

For Red Hat-based distributions:

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

# Step 2: Create AWS Credentials

## Create a Temporary User

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

IAM > Users > Create user

Step 1 Specify user details

Step 2 Set permissions

Step 3 Review and create

### Specify user details

#### User details

User name

CMPE583-GreenGrassUser

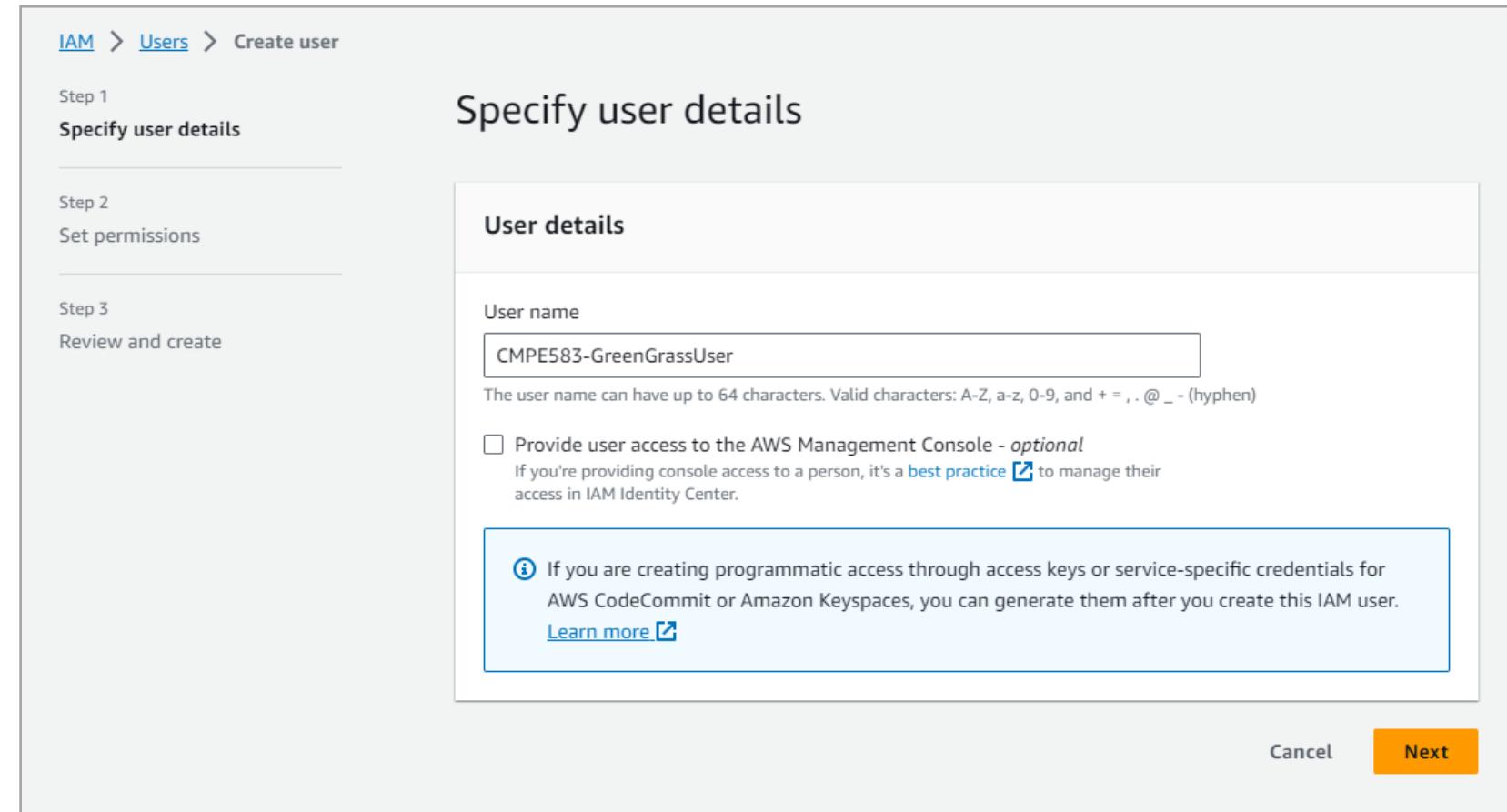
The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ \_ - (hyphen)

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

**ⓘ If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user.**

[Learn more](#)

Cancel Next



# Step 2: Create AWS Credentials

## Create Access Keys for the User

The screenshot shows the AWS IAM User Security Credentials page for the user 'CMPE583-GreenGrassUser'. The navigation path is IAM > Users > CMPE583-GreenGrassUser. The 'Security credentials' tab is selected. The 'Console sign-in' section shows a 'Console sign-in link' (https://arcelik-electronics.signin.aws.amazon.com/console) and a 'Console password' status of 'Not enabled'. The 'Access keys' section indicates '0' access keys. A large orange arrow points from the text 'Instead, use tools which provide short term credentials.' to the 'Create access key' button.

IAM > Users > CMPE583-GreenGrassUser

### CMPE583-GreenGrassUser [Info](#)

[Delete](#)

Permissions Groups Tags **Security credentials** Access Advisor

#### Console sign-in

Enable console access

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

Console password Not enabled

#### Access keys (0)

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

[Create access key](#)

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

[Create access key](#)

# Step 2: Create AWS Credentials

## Copy the Access Keys

IAM > Users > CMPE583-GreenGrassUser > Create access key

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

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

Step 3  
**Retrieve access keys**

### Retrieve access keys Info

**Access key**  
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.

Access key	Secret access key
<input type="text"/> AKIAWY5JNGIBCSFIMYHZ	<input type="text"/> ***** <a href="#">Show</a>

### Access key best practices

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

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

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

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

The screenshot shows the AWS IoT Greengrass Core devices interface. The left sidebar has a 'Core devices' section selected. The main area displays 'Greengrass core devices' with a count of 0. A large orange arrow points to the 'Set up one core device' button at the bottom of the list table.

AWS IoT > Greengrass > Core devices

## Greengrass core devices Info

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

Search by core device name

Name	Status	Status reported
No core devices You don't have any Greengrass core devices in eu-central-1.		

Set up one core device

Manage

- All devices
- Greengrass devices

Core devices

- Components
- Deployments
- Groups (V1)

- LPWAN devices
- Software packages Preview
- Remote actions
- Message routing
- Retained messages
- Security
- Fleet Hub

Device software

Billing groups

Settings

Feature spotlight

Frankfurt ▾ AD-System-Admin/AR430805@arcelik.com ▾

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

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

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

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

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

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

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

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

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

# Step 4: Install the Greengrass Core Software

## Configure AWS Credentials on the Client

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

Provide credentials as environment variables before running the installer:

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

- Credentials are not saved by the Greengrass installer!

# Step 4: Install the Greengrass Core Software For Linux Based Devices

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

Download the installer:

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

Run the installer:

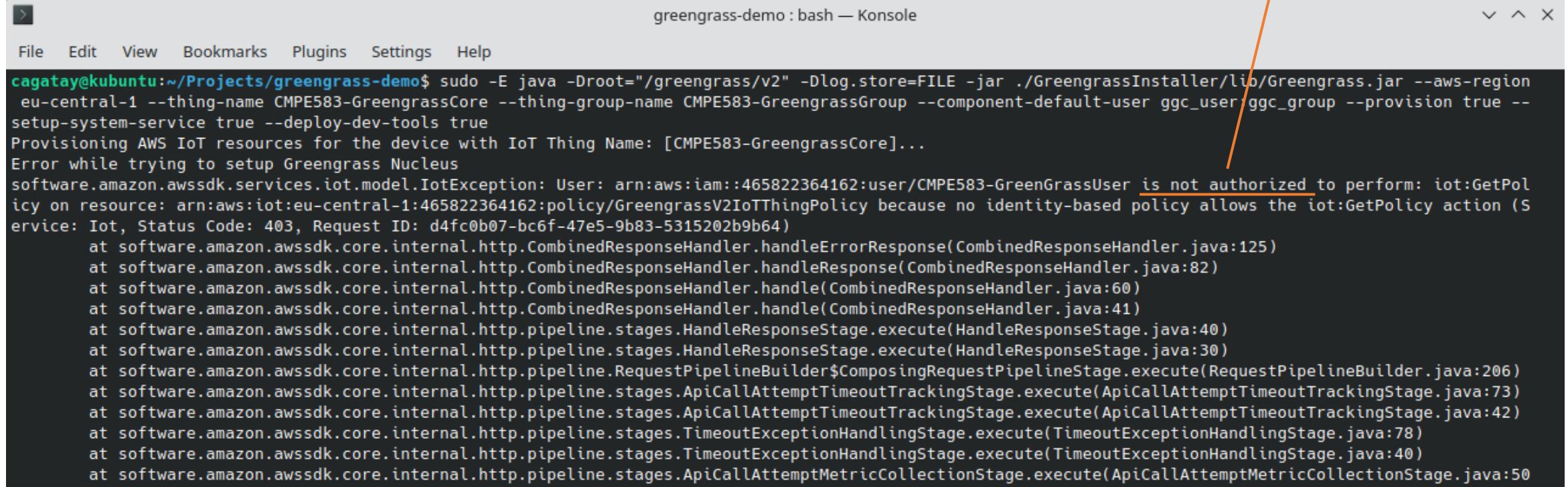
```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE -jar ./GreengrassInstaller/lib/Greengrass.jar --aws-  
region eu-central-1 --thing-name CMPE583-GreengrassCore --thing-group-name CMPE583-  
GreengrassGroup --component-default-user ggc_user:ggc_group --provision true --setup-system-service  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

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

# Step 4: Install the Greengrass Core Software

## Run the Installer Command

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



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

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

The command fails with the following error message:

```
Provisioning AWS IoT resources for the device with IoT Thing Name: [CMPE583-GreengrassCore]...
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)
```

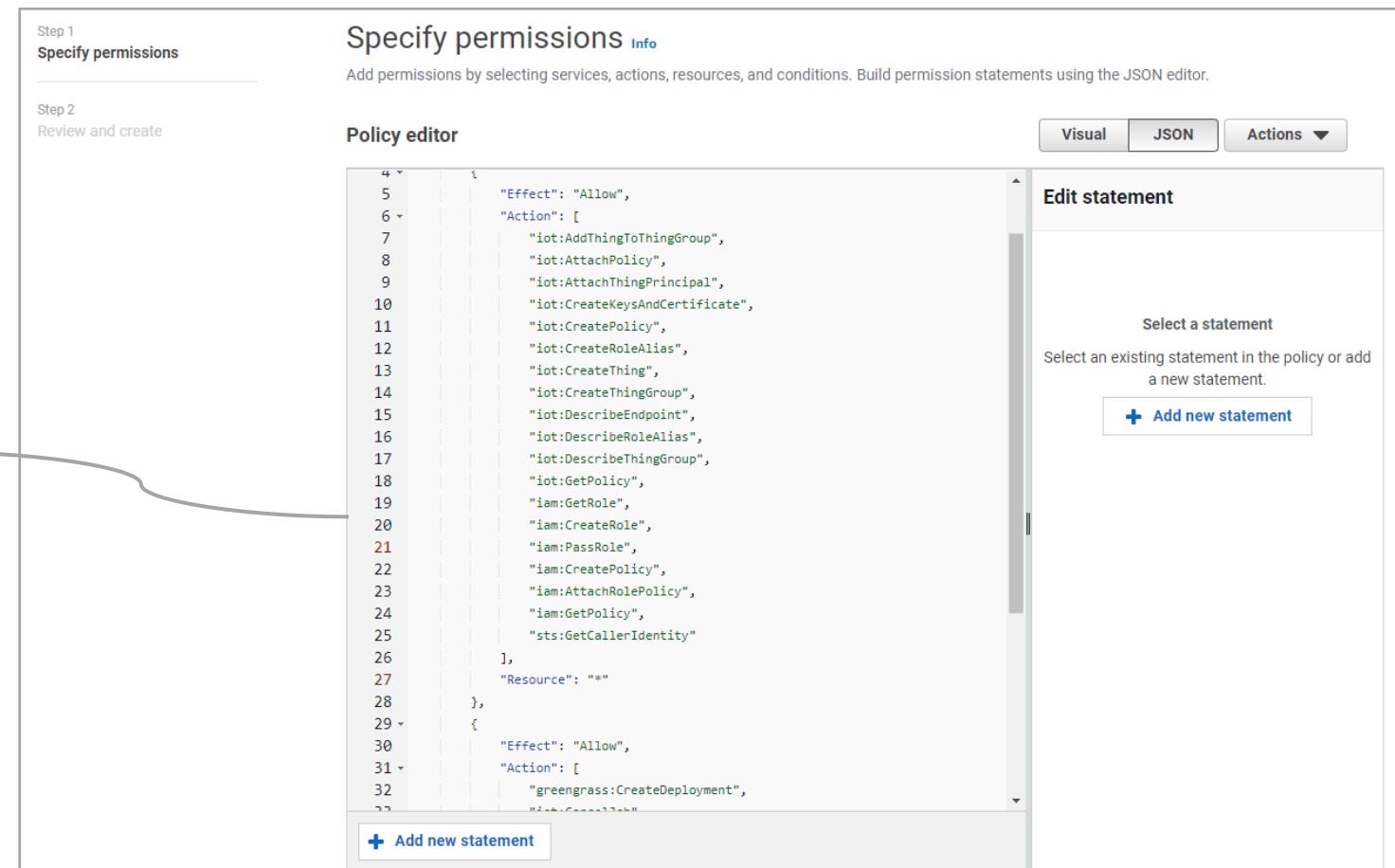
An orange arrow points from the text "The user is not authorized to perform installer jobs! You should attach required permissions!" to the error message in the terminal.

# 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.

- *AWS/IAM/cmpe583-GreengrassUser-ProvisionPolicy.json*



# Step 4: Install the Greengrass Core Software

Rerun the Installer and Check If Core Device Is Healthy

The screenshot shows the AWS IoT Greengrass Core devices interface. At the top left, there is a breadcrumb navigation: AWS IoT > Greengrass > Core devices. Below the breadcrumb, the title "Greengrass core devices" is displayed with a blue "Info" link next to it. On the right side of the title, there are three buttons: a white button with a "C" icon, a white button labeled "Configure cloud discovery", and an orange button labeled "Set up one core device". Below these buttons is a search bar with the placeholder text "Search by core device name". To the right of the search bar are navigation icons: a left arrow, the number "1", a right arrow, and a gear icon. The main content area has a table header with columns: "Name" and "Status". A single data row is shown below the header, containing the name "CMPE583-GreengrassCore" and the status "Healthy". To the right of the status column, it says "Status reported 5 minutes ago".

Name	Status	Status reported
<a href="#">CMPE583-GreengrassCore</a>	<span>Healthy</span>	5 minutes ago

# Step 4: Install the Greengrass Core Software

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

```
1 [ {  
2     "Version": "2012-10-17",  
3     "Statement": [  
4         {  
5             "Effect": "Allow",  
6             "Action": [  
7                 "logs:CreateLogGroup",  
8                 "logs:CreateLogStream",  
9                 "logs:PutLogEvents",  
10                "logs:DescribeLogStreams",  
11                "s3:GetBucketLocation"  
12            ],  
13            "Resource": "*"  
14        }  
15    ]  
16 }
```

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

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

# Step 4: Install the Greengrass Core software

## Troubleshooting

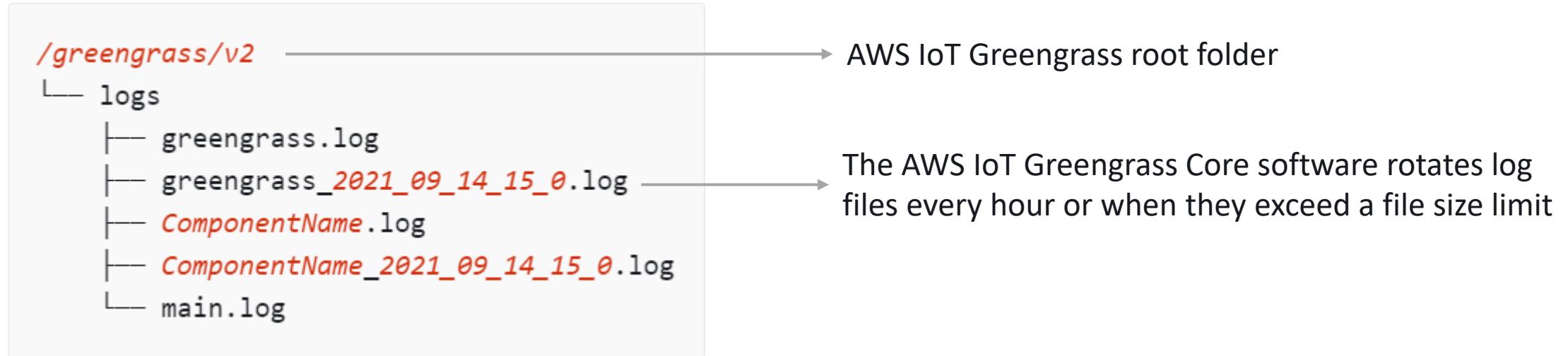
- If the GreengrassInstaller does not install the GreengrassV2TokenExchangeRole and prints an error message, you can create a custom role in IAM console with the following JSON value:

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

# Step 4: Install the Greengrass Core software

## Monitor AWS IoT Greengrass logs

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



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

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

# Step 4: Install the Greengrass Core Software

## Auto Start of Greengrass Core Software

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

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

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

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

# Uninstall the AWS IoT Greengrass Core Software For Linux Based Devices

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

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

- Verify that the service is deleted.

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

- Remove the root folder from the device.

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

# Greengrass Components

---

# Greengrass Components

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

## Recipe File

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

## Artifacts

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

# Step 1: Upload the Artifacts

## Create a bucket via AWS S3 (Console)

Amazon S3 > Buckets > Create bucket

### Create bucket Info

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

#### General configuration

Bucket name

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

AWS Region

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

[Choose bucket](#)

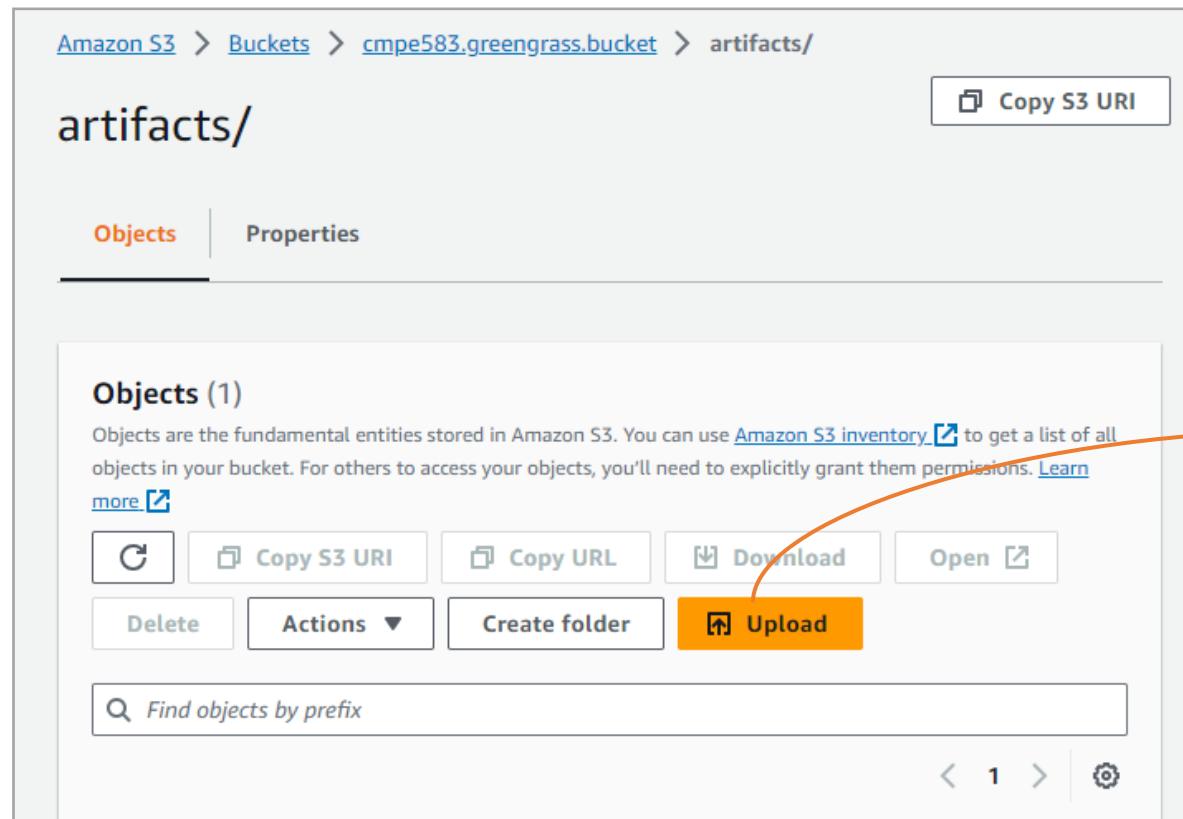


- Be sure that the S3 bucket is in the same AWS Region where you create the component.
- AWS IoT Greengrass doesn't support cross-Region requests for component artifacts!

# Step 2: Upload the Artifacts

Create artifacts folder & upload files

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



The code editor window displays a Python script named 'cmpe583-PrintMsg.py'. The code is as follows:

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

# Step 2: Set Policies to Access Files in S3

## Via AWS IAM (Console)

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

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

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

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

IAM > Roles > GreengrassV2TokenExchangeRole

## GreengrassV2TokenExchangeRole [Info](#)

Delete

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

Permissions    Trust relationships    Tags    Access Advisor    Revoke sessions

---

**Permissions policies (1) [Info](#)**

You can attach up to 10 managed policies.

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

[Attach policies](#)

[Create inline policy](#)

Filter by Type

[All types](#) ▾

< 1 > [⚙️](#)

<input type="checkbox"/>	Policy name <a href="#">Edit</a>	Type	Attached entities
<input type="checkbox"/>	<a href="#">GreengrassV2TokenExch...</a>	Customer managed	1

## Specify permissions Info

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

### Policy editor

VisualJSON

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

Edit statement

Select an existing statement  
Add new statement

# Step 3: Create the Component

Via AWS IoT Greengrass (Console)

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

# Step 3: Create the Component

## Define the Recipe

AWS IoT > Greengrass > Components > Create component

### Create component

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

#### Component information

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

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

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

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

#### Recipe

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

```
1 ▼ {  
2     "RecipeFormatVersion": "2020-01-25",  
3     "ComponentName": "samplePrintMsg",  
4     "ComponentVersion": "1.0.0",  
5     "ComponentDescription": "My test AWS IoT Greengrass component.",  
6     "ComponentPublisher": "Amazon",  
7     "ComponentConfiguration": {  
8         "DefaultConfiguration": {"Message": "custom test message"}  
9     },  
10    "Manifests": [  
11        {  
12            "Platform": { "os": "linux" },  
13            "ManifestType": "aws-iot-device-sdk"  
14        }  
15    ]  
16}
```

# Step 3: Create the Component

## Configure the Recipe

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

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

Define how to execute your artifact

Provide the artifact URI

# Step 4: Deploy the Component

## Via AWS IoT Greengrass (console)

AWS IoT > Greengrass > Components > samplePrintMsg:1.0.0

### samplePrintMsg

Version: 1.0.0 ▾ Create version Delete version View recipe Deploy

#### Overview

Description  
My test AWS IoT Greengrass component.

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

#### Default configuration

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

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



# Step 4: Deploy the Component

## Configure Deployment

AWS IoT > Greengrass > Deployments > Create deployment

Step 1  
**Specify target**

Step 2  
Select components

Step 3 - optional  
Configure components

Step 4 - optional  
Configure advanced settings

Step 5  
Review

**Specify target**

**Deployment information**

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

Test Deployment for CMPE583

The deployment name can have up to 255 characters.

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

Target type

Core device

Thing group

Target name

CMPE583-GreengrassGroup



# Step 4: Deploy the Component

## Select the Component

AWS IoT > Greengrass > Deployments > Create deployment

Step 1  
[Specify target](#)

Step 2  
**Select components**

Step 3 - optional  
[Configure components](#)

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

Step 5  
[Review](#)

### Select components

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

**My components (1)**

<input checked="" type="checkbox"/>	Name 
<input checked="" type="checkbox"/>	<a href="#">samplePrintMsg</a>



**Public components (47)**

<input type="checkbox"/>	Name 
<input type="checkbox"/>	<a href="#">aws.greengrass.SNS</a>
<input type="checkbox"/>	<a href="#">aws.greengrass.LegacySubscriptionRouter</a>
<input type="checkbox"/>	<a href="#">aws.greengrass.clientdevices.mqtt.Bridge</a>

# Step 4: Deploy the Component

## Configure Advanced Settings

AWS IoT > Greengrass > Deployments > Create deployment

Step 1  
[Specify target](#)

Step 2  
[Select components](#)

Step 3 - optional  
[Configure components](#)

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

Step 5  
Review

### Configure advanced settings - *optional*

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

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

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

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

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



Cancel   Previous   **Next**

# Step 4: Deploy the Component

## Review & Deploy

AWS IoT > Greengrass > Deployments > Create deployment

Step 1  
[Specify target](#)

Step 2  
[Select components](#)

Step 3 - optional  
[Configure components](#)

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

Step 5  
**Review**

### Review

#### Step 4: Advanced deployment configurations

[Edit](#)

##### Advanced deployment configurations

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

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

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

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

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

Failure handling policy  
Rollback

Deploy

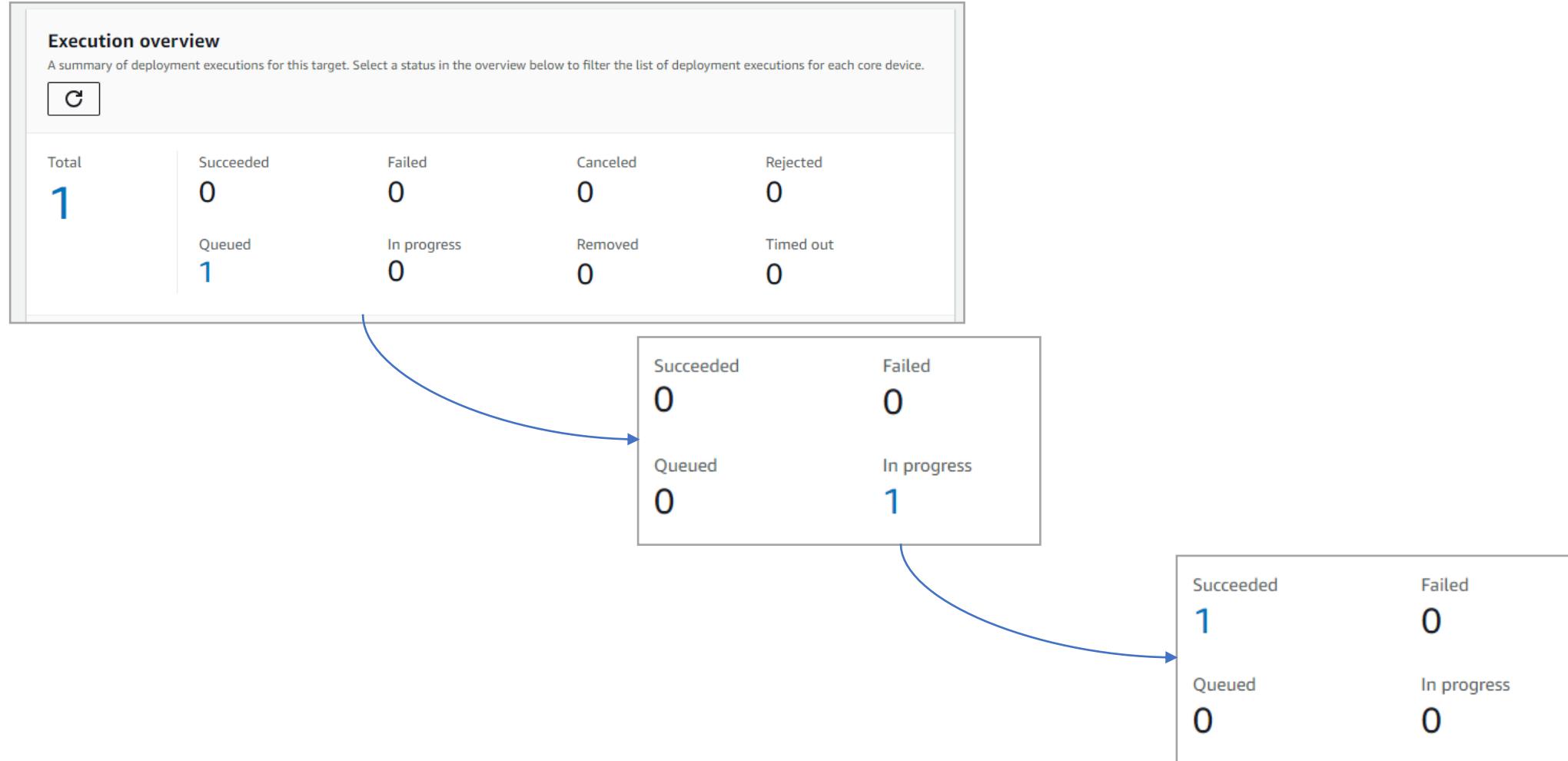
Download as JSON

Cancel

Previous

# Step 5: Verify the Component

## Check AWS IoT Greengrass Console



# Step 5: Verify the Component

## Check Log Files

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

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



All Good!

# Troubleshooting

AWS IoT > Greengrass > Deployments > Test Deployment for CMPE583

## Test Deployment for CMPE583

Latest revision: 1 ▾ Cancel Actions ▾

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

### Overview

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

Target <a href="#">CMPE583-GreengrassGroup</a>	Target type Thing group	Deployment created 1 minute ago
IoT job <a href="#">3ae1680c-7f4c-4068-86ea-88f9e4a4e038</a>	Deployment status <a href="#">Active</a>	

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> ←	Cancelled 0	Rejected 0
Queued 0	In progress 0	Removed 0	Timed out 0	

# Troubleshooting

## Check greengrass.log File

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

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



# AWS Provided Components

---

# AWS Provided Components

## Nucleus

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

# AWS Provided Components

## OS Support and More



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

# Collecting IP Addresses of Core Devices

## IPDetector

AWS IoT > Greengrass > Components

### Greengrass components Info

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

#### Public components (47)

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

Name	Publisher	Version	Architectures	Version created	Documentation
<a href="#">aws.greengrass.clientdevices.IPDetector</a>	AWS	2.1.7	All	3 months ago	<a href="#">Learn more</a> 

**Search bar:** ipde X 1 match < 1 > 



# Collecting IP Addresses of Core Devices

## Deploy an IPDetector Component

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

### aws.greengrass.clientdevices.IPDetector

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

**Overview** | View documentation

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

Version 2.1.7	Version created 3 months ago	Status <span style="color: green;">Deployable</span>	Component scope Public
Type aws.greengrass.plugin	ARN arn:aws:greengrass:eu-central-1:aws:components:aws.greengrass.clientdevices.IPDetector:versions:2.1.7	Publisher AWS	

### Dependencies (1)

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

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

# Collecting IP Addresses of Core Devices

## Add Public Component to Existing Deployment

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

### Add to deployment

Deployment

Add to existing deployment       Create new deployment

Find by deployment name or target name

Deployment	Target name	Target type	Status	Deployment created
<a href="#">Test Deployment for CMPE583</a>	CMPE583-GreengrassGroup	Thing group	Active	2 hours ago

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

# Collecting IP Addresses of Core Devices

## Select All Components You Want to Deploy

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

Step 1  
[Specify target](#)

Step 2 - optional  
[Select components](#)

Step 3 - optional  
[Configure components](#)

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

Step 5  
[Review](#)

### Select components - *optional*

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

**My components (1)**

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

**Public components (47)**

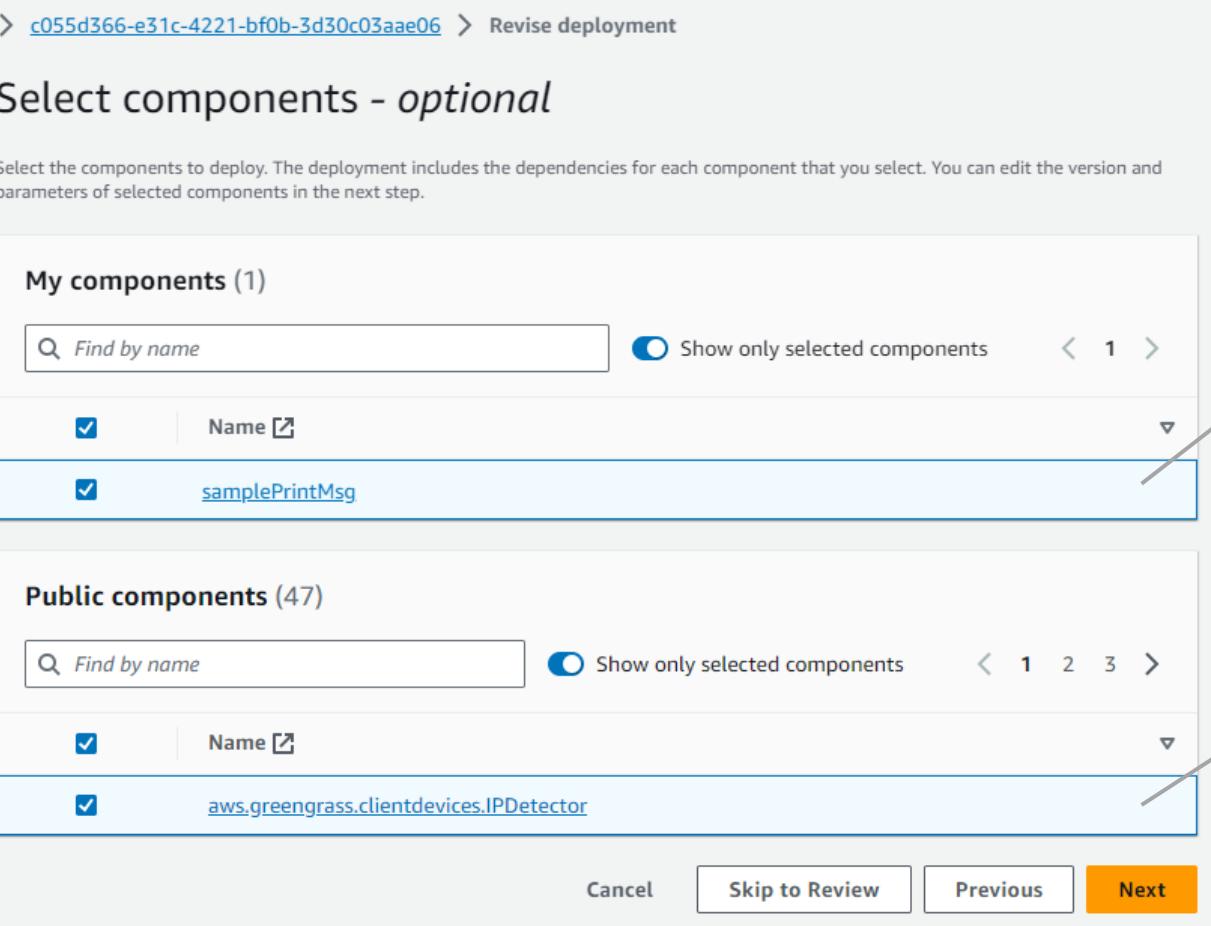
Name
<input checked="" type="checkbox"/> aws.greengrass.clientdevices.IPDetector

Show only selected components

Custom component

AWS Provided component

Cancel Skip to Review Previous Next



# Collecting IP Addresses of Core Devices

## Troubleshooting

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

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



# Troubleshooting

## Create an IAM Role

The screenshot shows the AWS Identity and Access Management (IAM) service interface. The left sidebar has a search bar and a navigation menu with sections like Dashboard, Access management, Roles, Policies, Identity providers, Account settings, Access reports, and others. The 'Roles' section is currently selected. The main content area is titled 'Roles (20)' and contains a brief description of what an IAM role is. It features a search bar, a 'Create role' button (which is highlighted with a large orange arrow), and a list of existing roles. Each role entry includes a checkbox, the role name, a description indicating it's a service role, and the corresponding AWS service. The list includes roles such as AWSServiceRoleForAccessAnalyzer, AWSServiceRoleForAmazonCodeGuruReviewer, AWSServiceRoleForAmazonSSM, AWSServiceRoleForCloudFormationStackSetsOrgMember, AWSServiceRoleForCloudTrail, AWSServiceRoleForElasticLoadBalancing, AWSServiceRoleForGlobalAccelerator, AWSServiceRoleForOrganizations, AWSServiceRoleForSSO, AWSServiceRoleForSupport, and AWSServiceRoleForTrustedAdvisor.

Role Name	Description
AWSServiceRoleForAccessAnalyzer	AWS Service: accessanalyzer
AWSServiceRoleForAmazonCodeGuruReviewer	AWS Service: codeguru
AWSServiceRoleForAmazonSSM	AWS Service: ssm (Systems Manager)
AWSServiceRoleForCloudFormationStackSetsOrgMember	AWS Service: member
AWSServiceRoleForCloudTrail	AWS Service: cloudtrail
AWSServiceRoleForElasticLoadBalancing	AWS Service: elasticloadbalancing
AWSServiceRoleForGlobalAccelerator	AWS Service: globalaccelerator
AWSServiceRoleForOrganizations	AWS Service: organizations
AWSServiceRoleForSSO	AWS Service: sso (Single Sign-On)
AWSServiceRoleForSupport	AWS Service: support
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor

# Troubleshooting

## Create an IAM Role

IAM > Roles > Create role

Step 1  
Select trusted entity

Step 2  
Add permissions

Step 3  
Name, review, and create

### Select trusted entity Info

**Trusted entity type**

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

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

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

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

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

**Service or use case**

Greengrass ▾

Choose a use case for the specified service.  
Use case

Greengrass  
Allows Greengrass to call AWS services on your behalf.

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

# Troubleshooting

## Add Permission to Created Role

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

Add permissions Info

Permissions policies (1/917) Info

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

Filter by Type

Q green

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

▶ Set permissions boundary - *optional*

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

# Troubleshooting

## Review and Create Role

IAM > Roles > Create role

Step 1  
[Select trusted entity](#)

Step 2  
[Add permissions](#)

Step 3  
**Name, review, and create**

### Name, review, and create

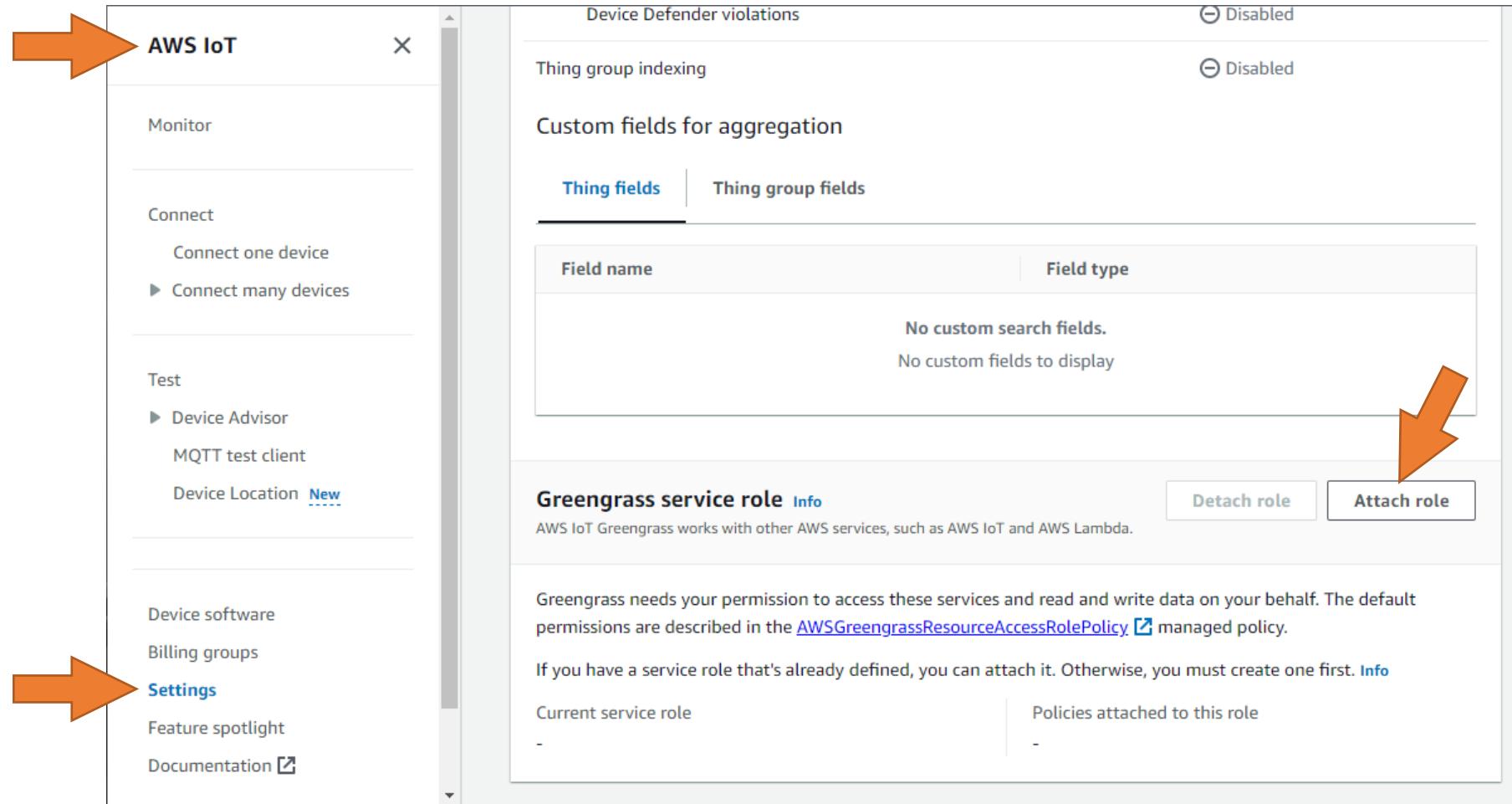
**Role details**

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

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

# Troubleshooting

## Attach Role to AWS IoT Service



# Troubleshooting

## Attach Role to AWS IoT Service

The screenshot shows the AWS IoT Greengrass service role configuration page. On the left, there's a sidebar with navigation links like Monitor, Connect, Test, Device software, Billing groups, Settings (which is selected), Feature spotlight, and Documentation. The main content area has tabs for Thing fields and Thing group fields, both currently disabled. It shows a table with Field name and Field type columns, stating "No custom search fields." and "No custom fields to display". Below this is a section for the Greengrass service role, which is currently attached. It includes a "Greengrass service role" button, a "Info" link, and "AWS IoT Greengrass works with other AWS services, such as AWS IoT and AWS Lambda." buttons for "Detach role" and "Change role". A note says "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." It also says "If you have a service role that's already defined, you can attach it. Otherwise, you must create one first." with an "Info" link. A success message box says "Greengrass service role has been updated successfully." At the bottom, it shows the current service role as "arn:aws:iam::465822364162:role/CMPE583-GreengrassRole" and the policies attached to it as "AWSGreengrassResourceAccessRolePolicy".

# Troubleshooting

## Verify the Deployment

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

The image shows two screenshots from the AWS IoT Greengrass console. The left screenshot is the 'Execution overview' page for a target, showing 1 deployment execution that has succeeded. An orange arrow points from the 'Succeeded' count on this page to the 'MQTT broker endpoints' section of the right screenshot. The right screenshot is the 'Core devices' page for a device named 'CMPE583-GreengrassCore'. It displays the 'Client devices' tab, which lists an MQTT broker endpoint with the endpoint '10.0.2.15' and port '8883'.

AWS IoT > Greengrass > Core devices > CMPE583-GreengrassCore

CMPE583-GreengrassCore

Components Deployments Thing groups Client devices Tags

MQTT broker endpoints (1) Info

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

Manage endpoints

Endpoint	Port	Connection metadata
10.0.2.15	8883	

# Publish/Subscribe AWS IoT Core MQTT Messages

# AWS IoT Greengrass Core IPC

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

# AWS IoT Core MQTT Messaging IPC

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

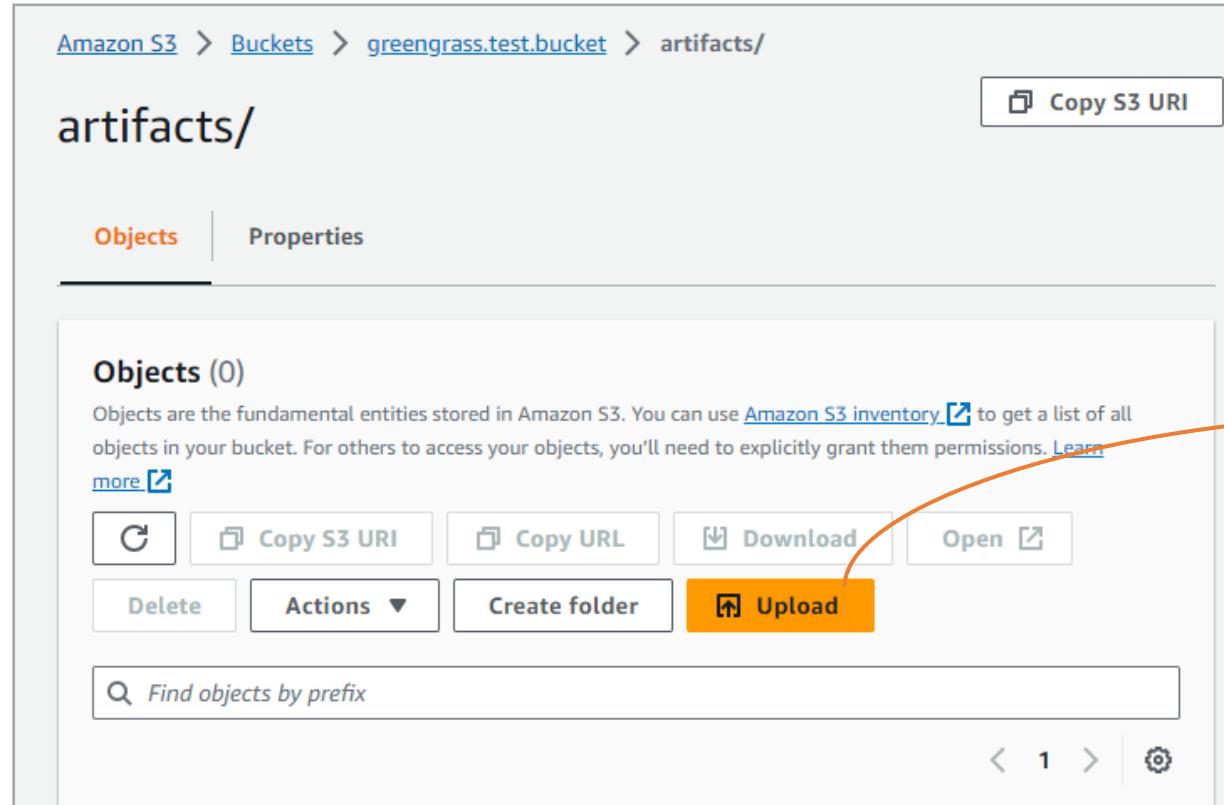
# MQTT

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

# Step 1: Develop & Upload Client-Side Code

## Via AWS IoT Greengrass (Console)

- Upload your artifacts to a folder in your S3 bucket



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

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

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

ipc_client = awsiot.greengrasscoreipc.connect()
...  
...
```

# Step 2: Create a Custom Component

## Via AWS IoT Greengrass (Console)

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

# Step 2: Create a Custom Component

## Enter the Recipe

AWS IoT > Greengrass > Components > Create component

### Create component

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

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

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

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

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

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

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

# Step 2: Create a Custom Component

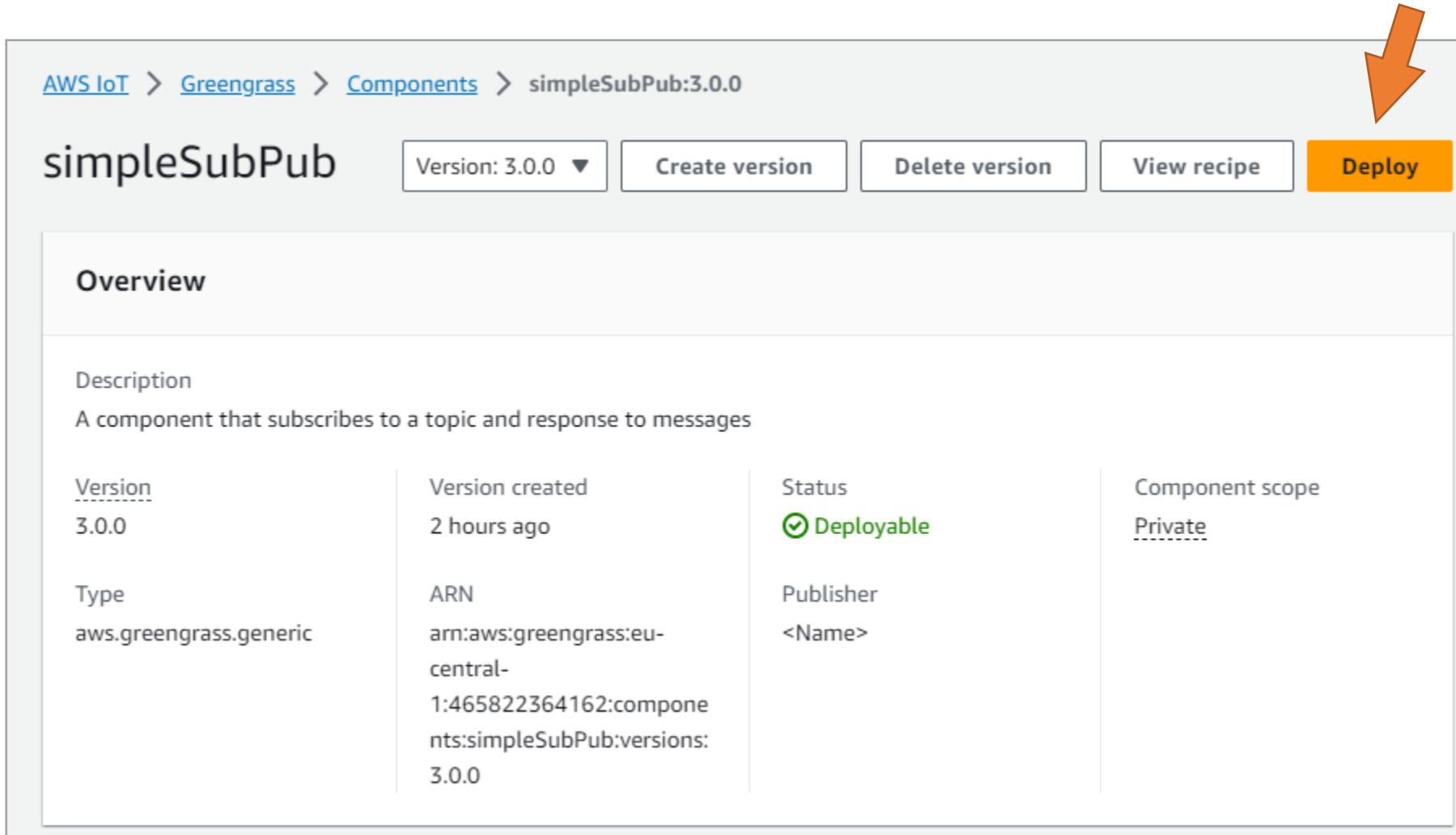
## Adjust the Access Control

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

The diagram shows two callout boxes pointing from the 'resources' field in the JSON code. The top callout contains the text: 'Allow the component to publish messages to AWS IoT Core or subscribe to messages from AWS IoT Core.' The bottom callout contains the text: 'A topic string, such as test/topic, or \* to allow access to all topics. You can use MQTT topic wildcards (# and +) to match multiple resources.'

# Step 3: Deploy Your Component

## Via AWS IoT Greengrass (Console)



AWS IoT > Greengrass > Components > simpleSubPub:3.0.0

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

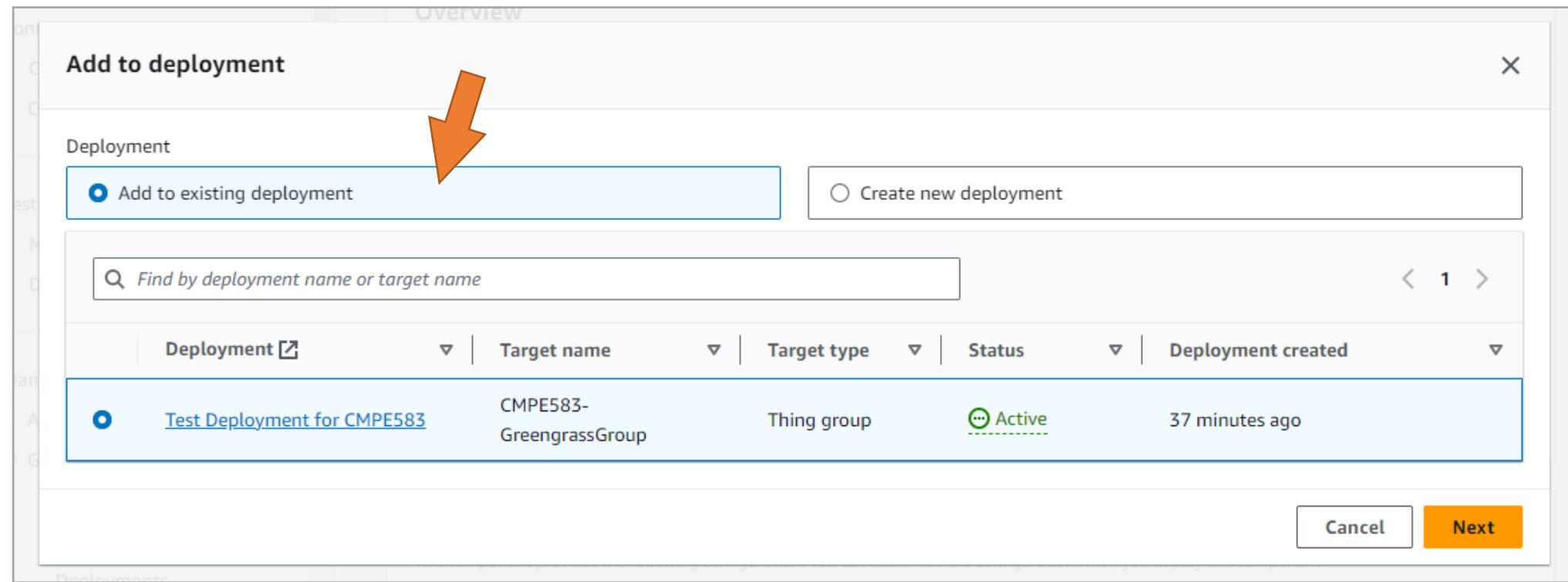
**Overview**

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

Version	Version created	Status	Component scope
3.0.0	2 hours ago	✓ Deployable	Private
Type	ARN	Publisher	
aws.greengrass.generic	arn:aws:greengrass:eu-central-1:465822364162:components:simpleSubPub:versions:3.0.0	<Name>	

# Step 3: Deploy Your Component

## Add Custom Component to Existing Deployment



# Step 3: Deploy Your Component

## Select the Component

**Select components - optional**

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

**My components (2)**

<input type="checkbox"/>	Name
<input checked="" type="checkbox"/>	<a href="#">samplePubSub</a>
<input checked="" type="checkbox"/>	<a href="#">samplePrintMsg</a>

**Public components (47)**

<input type="checkbox"/>	Name
<input checked="" type="checkbox"/>	<a href="#">aws.greengrass.clientdevices.IPDetector</a>

Cancel  Skip to Review  Previous  Next



# Step 3: Deploy Your Component

## Configure Advanced Settings

AWS IoT > Greengrass > Deployments > Create deployment

Step 1  
[Specify target](#)

Step 2  
[Select components](#)

Step 3 - optional  
[Configure components](#)

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

Step 5  
Review

### Configure advanced settings - *optional*

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

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

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

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

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



Cancel   Previous   **Next**

# Step 3: Deploy Your Component

## Review & Deploy

AWS IoT > Greengrass > Deployments > Create deployment

Step 1  
[Specify target](#)

Step 2  
[Select components](#)

Step 3 - optional  
[Configure components](#)

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

Step 5  
**Review**

### Review

#### Step 4: Advanced deployment configurations

[Edit](#)

#### Advanced deployment configurations

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

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

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

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

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

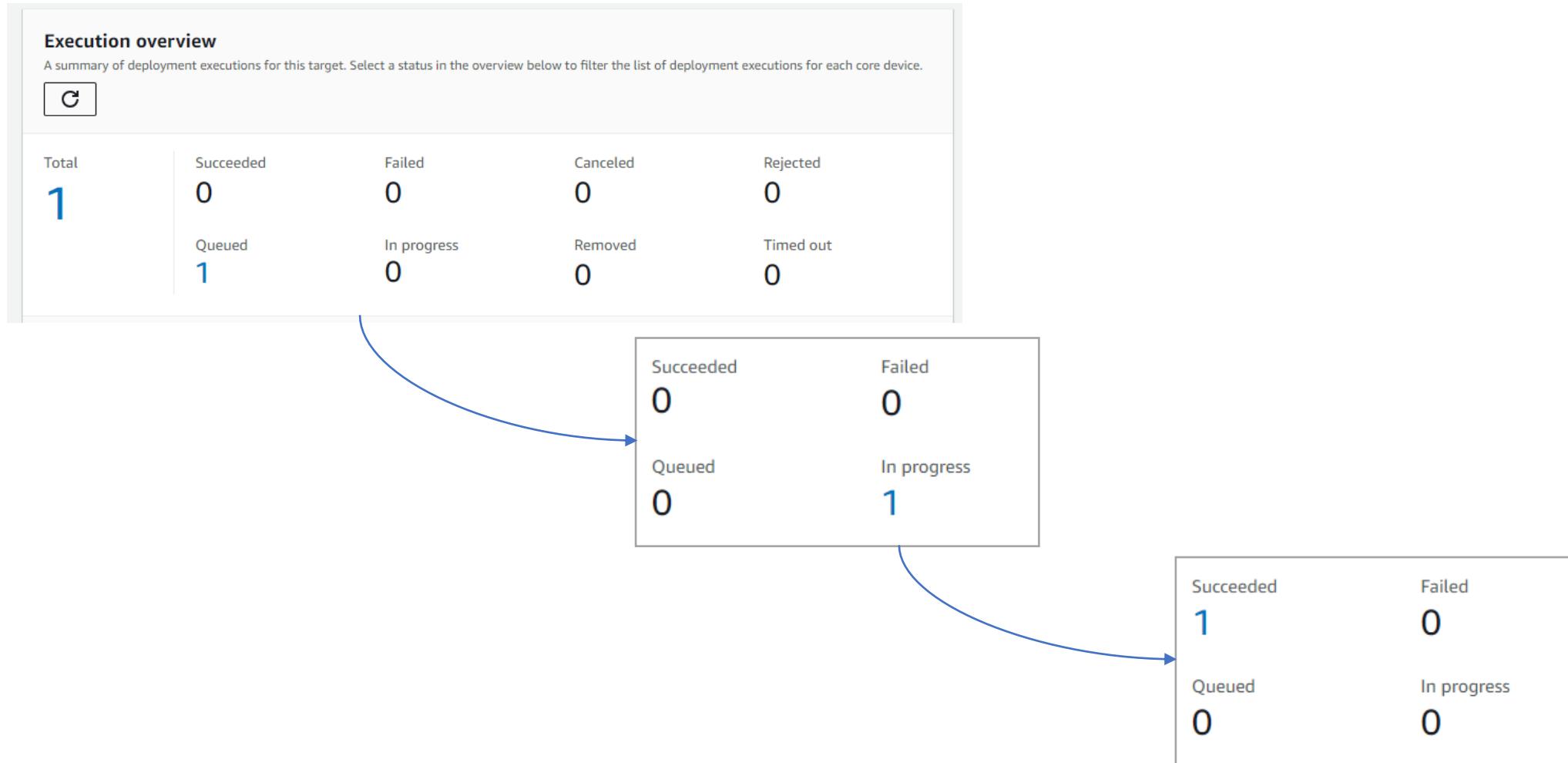
Failure handling policy  
Rollback



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

# Step 4: Verify the Component

## Check AWS IoT Greengrass Console



# Troubleshooting

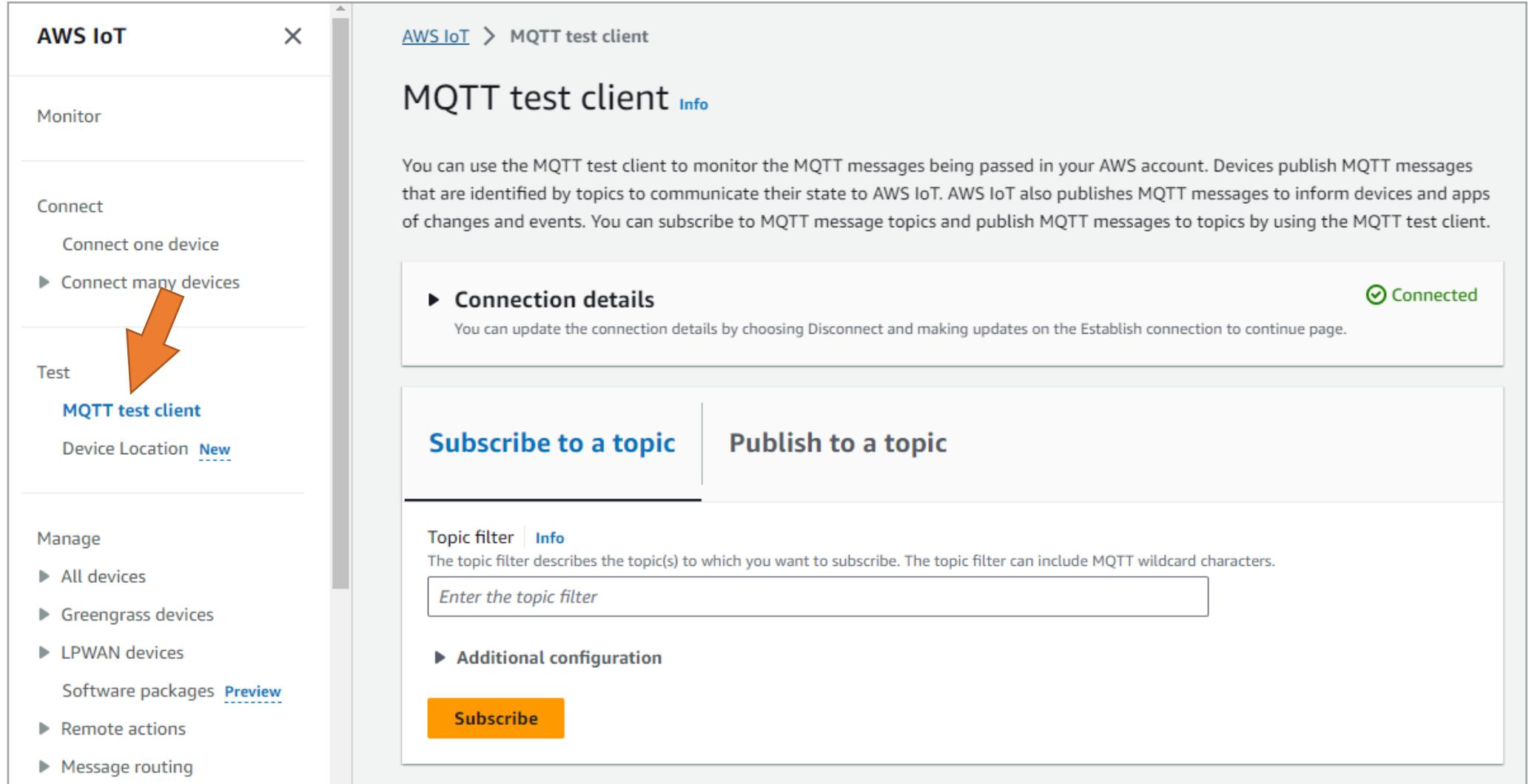
## Check greengrass.log File

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

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

# Test Your Deployment

## Via AWS MQTT Test Client



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

AWS IoT > MQTT test client

## MQTT test client Info

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

**Connection details** Connected

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

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

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

**Additional configuration**

**Subscribe**

# Test Your Deployment

## Subscribe All Topics

**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**

**Subscriptions** #

Pause Clear Export Edit

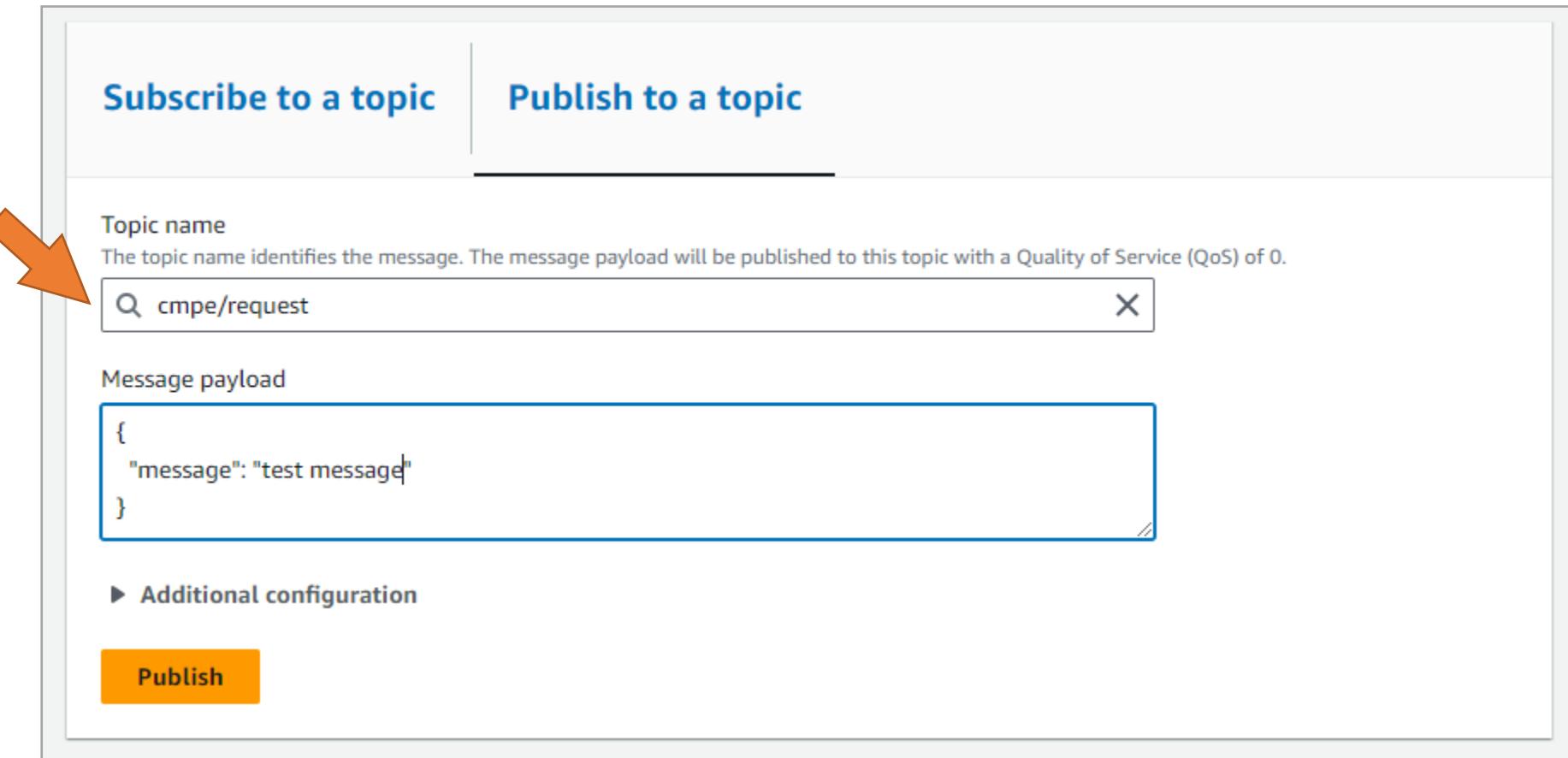
#	Heart	X
#	Heart	X

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

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

# Test Your Deployment

## Send a Test Message



The screenshot shows a user interface for publishing a message to an MQTT topic. At the top, there are two buttons: "Subscribe to a topic" and "Publish to a topic". The "Publish to a topic" button is highlighted with a thick black border. Below these buttons, there is a section labeled "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." A search bar contains the topic name "cmpe/request". An orange arrow points to the left side of the search bar. Below the topic name, there is a "Message payload" section containing a JSON object: { "message": "test message" }. At the bottom of the interface, there is a "▶ Additional configuration" link and a prominent orange "Publish" button.

Subscribe to a topic    Publish to a topic

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

cmpe/request

Message payload

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

▶ Additional configuration

Publish

# Test Your Deployment

## Check If the Client Responds Back

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

The main area displays two message entries:

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

```
{  "timestamp": 1698589146478,  "response": "Hello from the Green Grass Core Device!"}
```

A red arrow points from the "Publish" button towards this message entry.
- cmpe/request**: This entry was received on October 29, 2023, at 17:19:05 (UTC+0300). It contains the following JSON payload:

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

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

# Other Use-Cases of AWS IoT Greengrass Core IPC

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

# AWS Lambda Functions on Core Devices

# Run AWS Lambda Functions on Core Devices

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

# Lambda Function Requirements

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

# Lambda Function Lifecycle

## On-demand functions

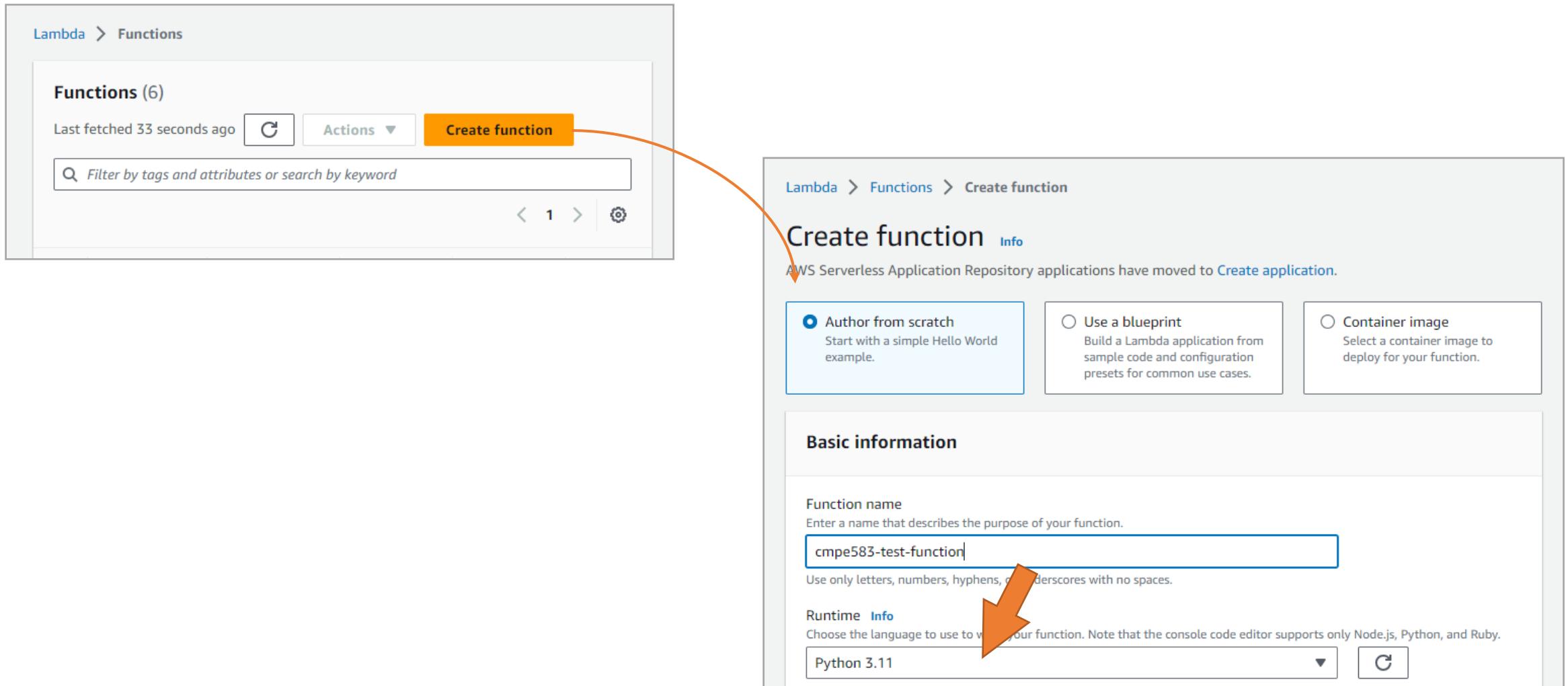
- On-demand functions start when they are invoked and stop when there are no tasks left to run.
- Each invocation of the function creates a separate container, also called a sandbox, to process invocations, unless an existing container is available for reuse.
- Any of the containers might process data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

## Long-lived functions

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

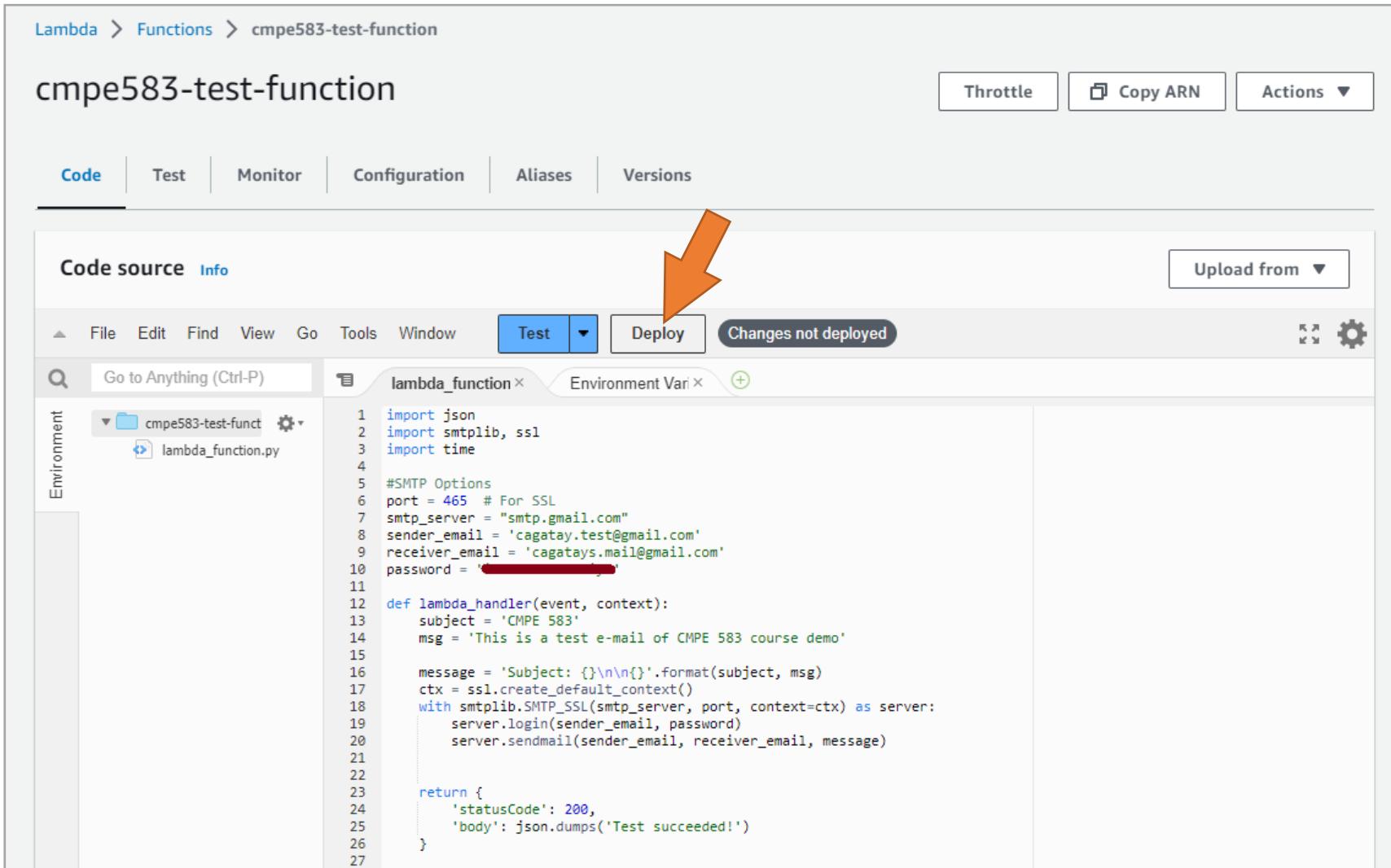
# Step 1: Create AWS Lambda Function

## Via AWS Lambda (Console)



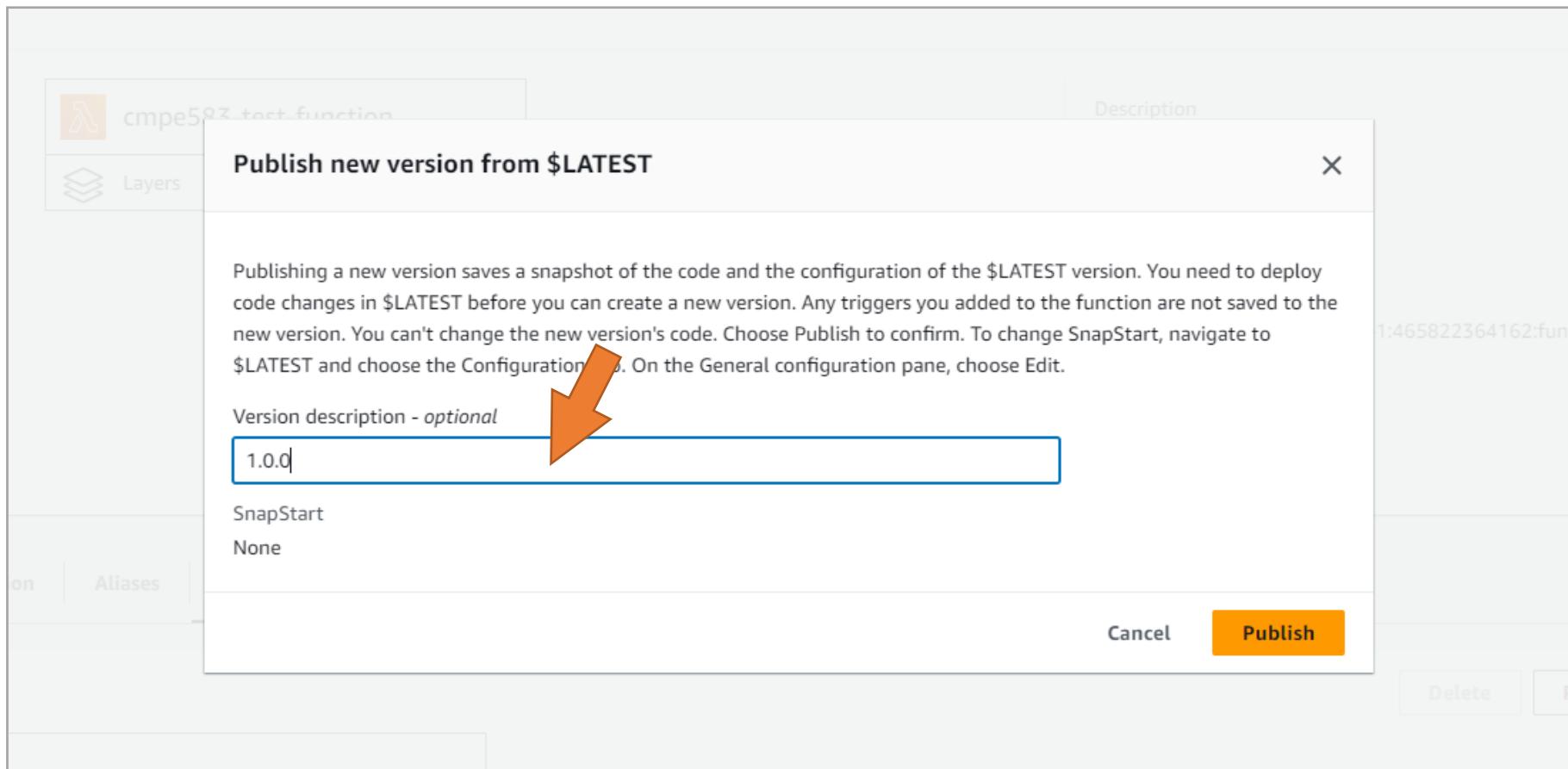
# Step 1: Create AWS Lambda Function

## Deploy the Lambda Function



# Step 1: Create AWS Lambda Function

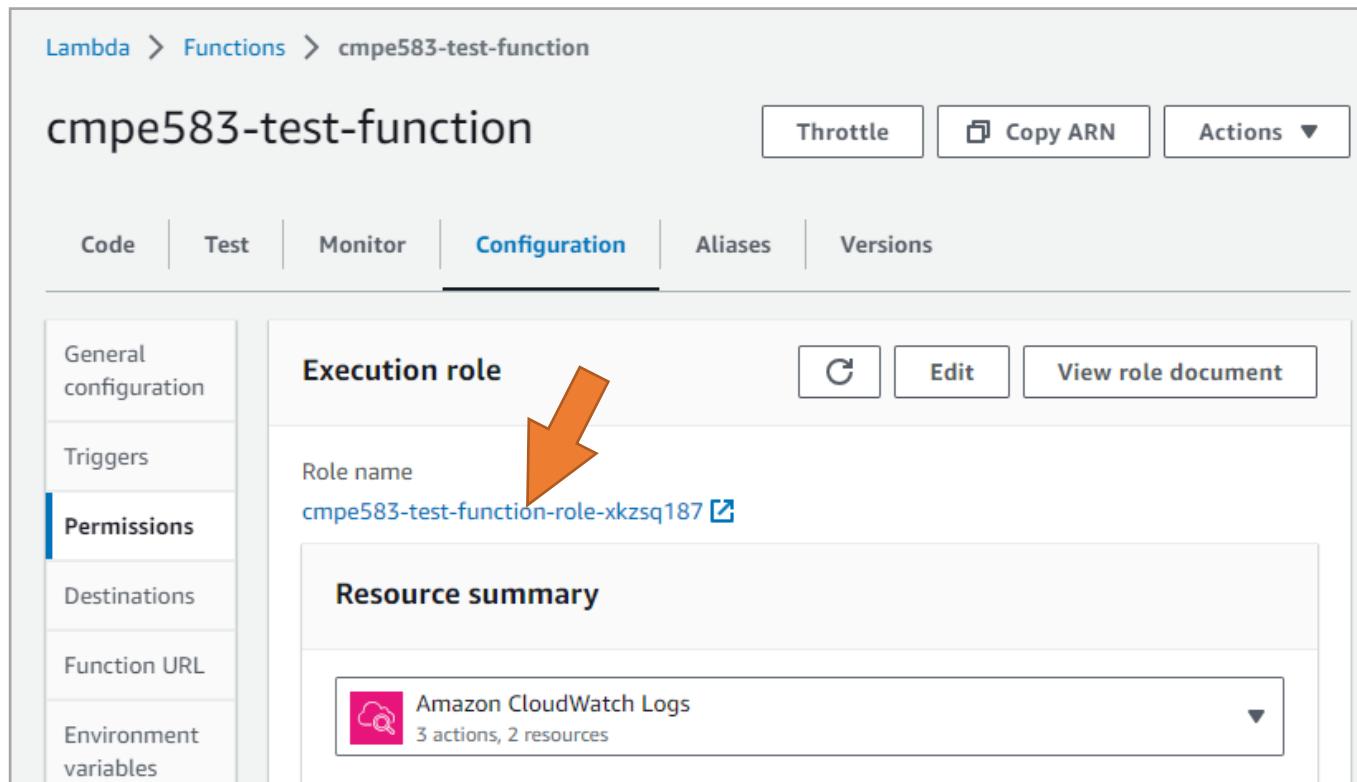
## Publish the First Version



# Step 1: Create AWS Lambda Function

## Configure Lambda Role

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



# Step 2: Create a Custom Component

## Import Lambda Function

The image shows two screenshots of the AWS IoT Greengrass Components interface. The left screenshot is titled 'Greengrass components' and shows a list of 'My components' (1). It includes a 'Create component' button, which is highlighted with an orange box and an arrow pointing to it from the first screenshot. The right screenshot is titled 'Create component' and provides instructions for creating a component. It features three options for component source: 'Enter recipe as JSON', 'Enter recipe as YAML', and 'Import Lambda function'. The 'Import Lambda function' option is selected and highlighted with a blue box and an orange arrow pointing to it from the first screenshot. Both screenshots include standard navigation elements like back/forward buttons and search bars.

AWS IoT > Greengrass > Components

### Greengrass components Info

My components Public components Community components

**My components (1)**  
Your components are private components that only you can see and deploy to core devices. [Learn more](#)

< 1 >

AWS IoT > Greengrass > Components > Create component

### Create component

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

#### Component information

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

#### Component source

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

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

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

#### Lambda function

Choose a Lambda function to import. [View all functions in the AWS Lambda console](#)

cmpe583-test-function python3.11 ▾

#### Lambda function version

Choose the version of the function to import.

Choose a version ▾

#### Component name - optional

A friendly name lets you identify this component. Defaults to the name of the Lambda function.

cmpe583.lambda.test

The name can be up to 128 characters. Valid characters: a-z, A-Z, 0-9, period (.), underscore (\_), and hyphen (-)

# Step 2: Create a Custom Component

## Configure Event Source to Trigger Lambda Function

**Lambda function configuration - optional**

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

**Event sources**

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

Topic	Type	Remove
cmpe/lambda	AWS IoT Core MQTT	

**Add event source**

**Timeout (seconds)**

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

3

The timeout must be a positive integer.

**Pinned**

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

True  
 False

► Additional parameters



# Step 2: Create a Custom Component

## Configure Lambda Lifecycle

The screenshot shows the 'Create component' dialog for a Lambda function. It includes two sections: 'Lambda function configuration' and 'Linux process configuration - optional'.

**Lambda function configuration:**

- Timeout (seconds):** A text input field containing '3'. An orange arrow points to this field.
- Pinned:** A section with a description and two radio button options: 'True' (unselected) and 'False' (selected).
- Additional parameters:** A section with a disclosure arrow.

**Linux process configuration - optional:**

- Isolation mode:** A dropdown menu currently set to 'No container'. An orange arrow points to this dropdown.

**Buttons at the bottom:**

- Cancel
- Create component

# Step 3: Deploy Your Component

## Add Component to Existing Deployment

The screenshot shows the AWS IoT Greengrass Components interface. On the left, a component named "cmpe583.lambda.test" is selected, with its version set to 1.0.0. The "Deploy" button is highlighted in orange. A curved arrow points from this button to the "Add to deployment" dialog box on the right.

**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</a>	CMPE583-GreengrassGroup	Thing group	Active

Cancel    Next

# Step 3: Deploy Your Component

## Configure Component to Merge Recipe

**Configure components - optional**

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

**Selected components (4)**

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

**Configure component**

**Find by name**

**Next**

**Configuration to merge**

The configuration to merge with the configuration on each core device. The deployment merges this JSON object after it resets the values that you specify in the list of reset paths. [Learn more](#)

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

# Test Your Deployment

## Via AWS MQTT Test Client

AWS IoT > MQTT test client

### MQTT test client Info

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

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

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

**Message payload**  
{  
  "message": "test"  
}  
  
▶ Additional configuration

**Publish**

Gmail

Oluştur

Gelen Kutusu 41

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

Postalarda arayın

CMPE 583 Gelen Kutusu

cagatay.test@gmail.com Alıcı: ▾  
İngilizce ▾ > Türkçe ▾ İletiyi çevir

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

Yanıtla Yönleendir

**All Good!**

# Test Your Deployment

## Verify From The Core Device's Logs

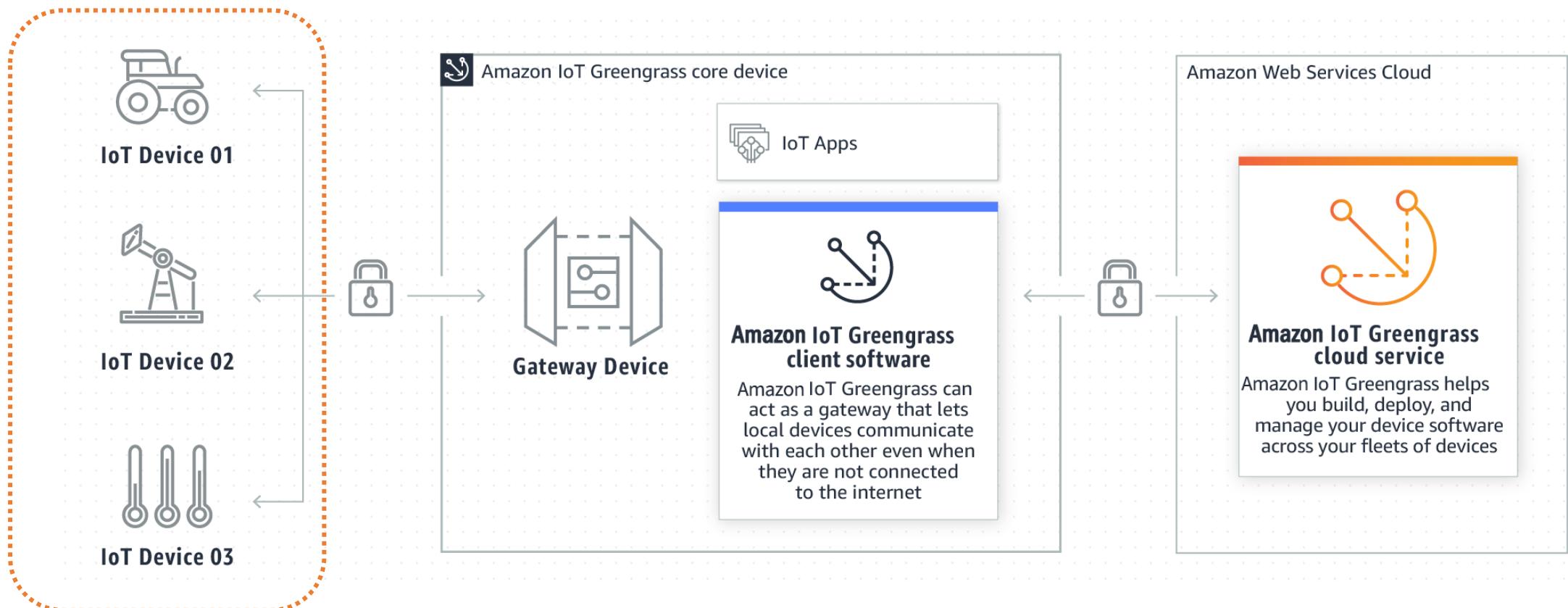
```
root@kubuntu:/home/cagatay/Projects/greengrass-demo# tail -n 100 -f /greengrass/v2/logs/cmpe583.lamda.test.log
2023-10-29T14:51:24.654Z [INFO] (pool-2-thread-52) cmpe583.lamda.test: shell-runner-start. {scriptName=services.cmpe583.lamda.test.lifecycle.startup.script, serviceInstance=1, serviceName=cmpe583.lamda.test,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



# Edge Computing with AWS IoT Greengrass - Part #2

# Local IoT Devices

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



# Client Device and Core Device Communication

## Use -Cases

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

# AWS-Provided Client Device Components

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

# Creating Client Devices

---

# Step 1: Configure the Greengrass Service Role

## Verify If Greengrass Service Role is Attached

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

The screenshot shows the AWS IoT Greengrass service role configuration page. On the left, there is a sidebar with links: Device software, Billing groups, Settings (which is selected), Feature spotlight, and Documentation. At the bottom of the sidebar is a link to 'Tell us what you think'. The main content area has a header 'Greengrass service role' with an 'Info' link, a 'Detach role' button, and a 'Change role' button. Below the header, a note states: 'AWS IoT Greengrass works with other AWS services, such as AWS IoT and AWS Lambda.' A large text block explains that Greengrass needs permission to access services and read/write data, mentioning the 'AWSGreengrassResourceAccessRolePolicy' managed policy. It also notes that a service role can be attached or created. At the bottom, it shows the current service role as 'arn:aws:iam::465822364162:role/CMPE583-GreengrassRole' and the policies attached to it as 'AWSGreengrassResourceAccessRolePolicy'.

AWS IoT

Device software

Billing groups

**Settings**

Feature spotlight

Documentation ↗

Tell us what you think

Greengrass service role [Info](#)

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](#)

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

Policies attached to this role  
AWSGreengrassResourceAccessRolePolicy

# Step 2: Configure the AWS IoT Thing Policy

Via AWS IoT Console

The screenshot shows the AWS IoT Console interface. A navigation pane on the left lists options like Monitor, Connect, Test, and Manage. Under Manage, the 'Things' option is selected, indicated by an orange arrow labeled '2'. The main content area shows a 'Thing details' card for a thing named 'CMPE583-GreengrassCore', with an ARN listed below it. An orange arrow labeled '3' points to the thing name. Below the card is a tab bar with 'Certificates' selected, indicated by an orange arrow labeled '4'. The 'Certificates' section displays one attached certificate with a Certificate ID of 'c62e9f5e5a6932223537f9fb3c477368dd8f379f45a930bf7b0285bb91ffc3ac' and a status of 'Active'. An orange arrow labeled '5' points to the Certificate ID.

1 AWS IoT

2 Things

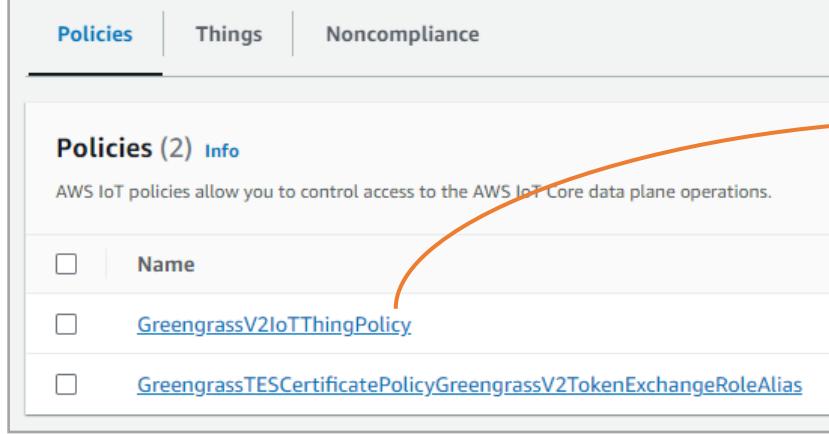
3 CMPE583-GreengrassCore

4 Certificates

5 c62e9f5e5a6932223537f9fb3c477368dd8f379f45a930bf7b0285bb91ffc3ac Active

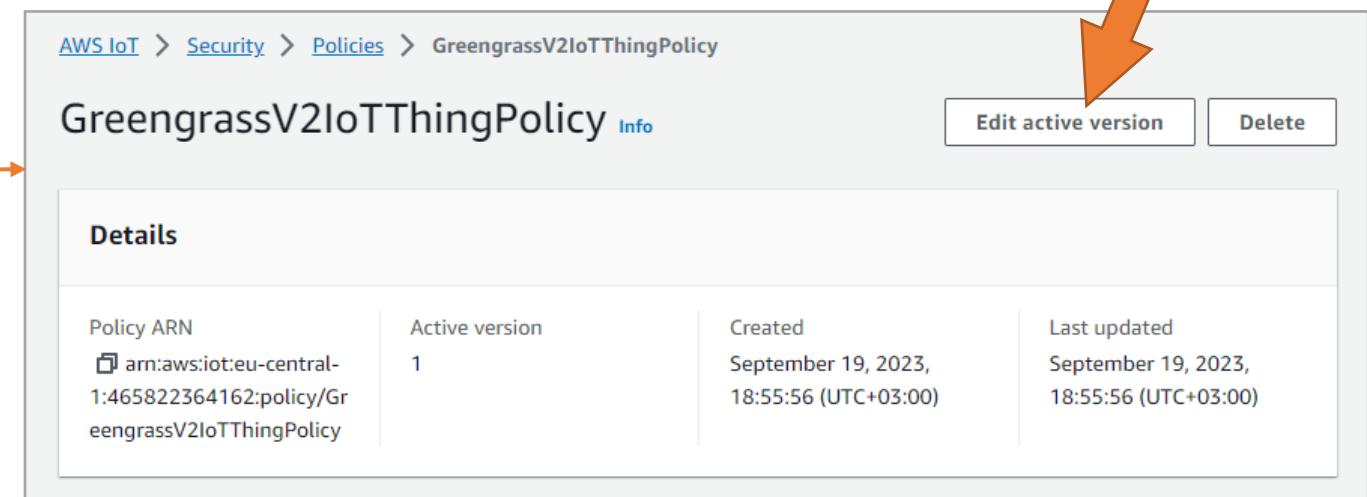
# Step 2: Configure the AWS IoT Thing Policy

## Review the Policy



AWS IoT Policies list:

- Name: GreengrassV2IoTThingPolicy
- Name: GreengrassTESCertificatePolicyGreengrassV2TokenExchangeRoleAlias



AWS IoT Policies > Security > Policies > GreengrassV2IoTThingPolicy

**GreengrassV2IoTThingPolicy** Info

**Details**

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

**Edit active version** **Delete**

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

# Step 2: Configure the AWS IoT Thing Policy

## Allow Core Device to Sync Shadows

AWS IoT > Security > Policies > GreengrassV2IoTThingPolicy > Edit

### Edit policy: GreengrassV2IoTThingPolicy (Version 1)

Policy statements | Policy examples

**Policy document** Info

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

Policy document

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

Builder JSON

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

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

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

# Step 2: Configure the AWS IoT Thing Policy

## Update & Verify Policy

All versions (1/2) [Info](#)

[C](#) [Delete](#) [Set as active](#) [Edit version](#) [View JSON](#)

The active and previous versions of this policy. Only one version can be active. A policy can have no more than 5 versions. To update a policy with 5 versions, you must first delete one.

Version number	Status	Created
<input checked="" type="checkbox"/> 2	<a href="#">Inactive</a>	November 04, 2023, 14:00:00
<input type="checkbox"/> 1	<a href="#">Active</a>	September 19, 2023, 18:00:00

Active version: 2 [Info](#)

[Builder](#) [JSON](#)

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

# Step 3: Create Client Device (Thing)

Via AWS IoT Console

The screenshot shows the AWS IoT Things management interface. On the left, a sidebar menu lists various device-related categories: Manage, All devices, Things (highlighted with an orange arrow), Thing groups, Thing types, Fleet metrics, Greengrass devices (also highlighted with an orange arrow), Core devices, Components, Deployments, and Groups (V1). The main content area is titled "AWS IoT > Manage > Things". It displays a summary section for "Things (2)" with an info link, followed by a search bar and navigation controls. Below this is a table listing two things: "CMPE583-GreengrassCore" and "MyFirstIOTThing". The "Create things" button, located at the top right of the table area, is also highlighted with an orange arrow.

Name	Thing type
CMPE583-GreengrassCore	-
MyFirstIOTThing	-

# Step 3: Create Client Device (Thing)

Via AWS IoT Console

AWS IoT > Manage > Things > Create things

## Create things Info

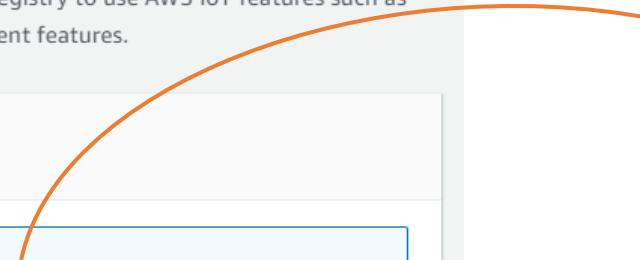
A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

**Number of things to create**

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

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

Cancel **Next**



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

Step 1 of 3

## Specify thing properties Info

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

**Thing properties Info**

**Thing name**

CMPE\_Test\_Client1

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

# Step 3: Create Client Device (Thing)

## Configure Certificate & Policy

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

Step 2 of 3

### Configure device certificate - optional Info

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

**Device certificate**

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

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

Step 3 of 3

### Attach policies to certificate - optional Info

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

**Policies (1/4)**

Select up to 10 policies to attach to this certificate.

Filter policies

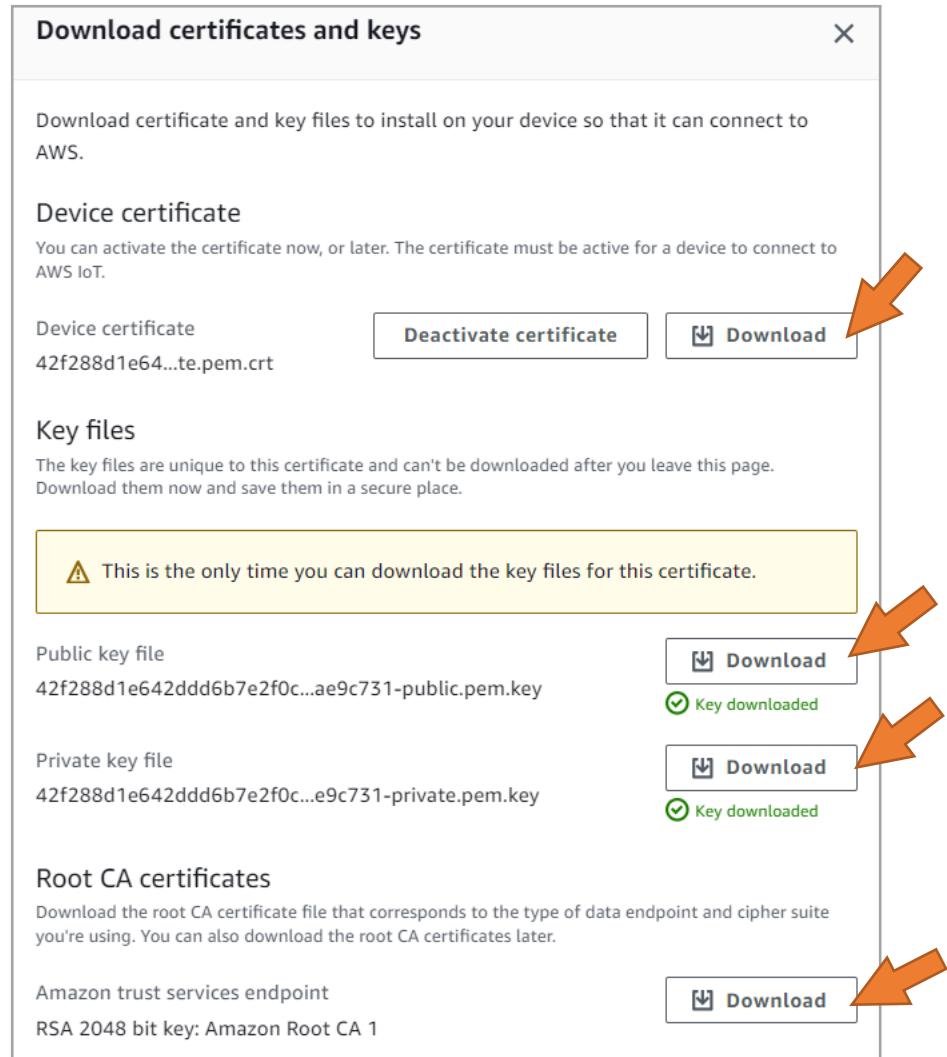
< 1 > Settings

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

Cancel Previous **Create thing**

# Step 3: Create Client Device (Thing)

## Download Key Files



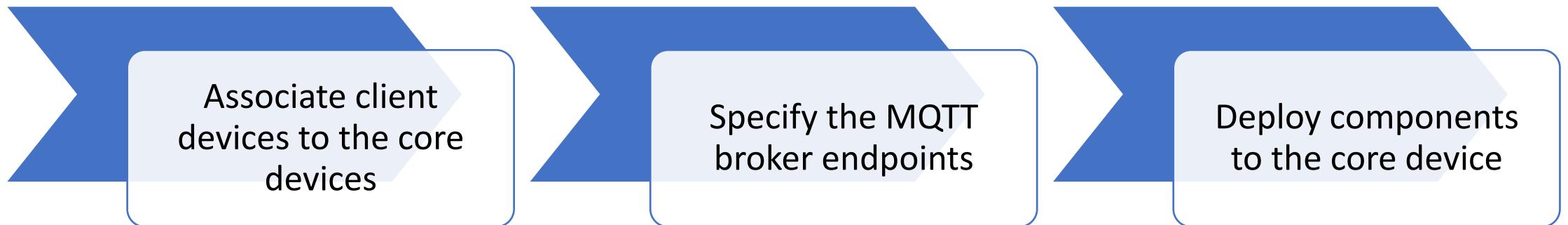
★ Key files must be downloaded in the last step of things creation!

# 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 configuration interface. On the left, a sidebar menu under 'Manage' includes 'All devices' (Things, Thing groups, Thing types, Fleet metrics) and 'Greengrass devices' (Core devices, Components, Deployments, Groups (V1)). The 'Core devices' option is selected. The main content area shows the title 'Greengrass core devices' with a count of '(1)'. Below it is a table with columns 'Name' and 'Status'. A single row is listed: 'CMPE583-GreengrassCore' with a status of 'Healthy' (indicated by a green checkmark icon). To the right of the table are buttons for 'Configure cloud discovery' (highlighted with an orange arrow), 'Set up one core device', and navigation controls (< 1 >). The breadcrumb navigation at the top indicates the path: AWS IoT > Greengrass > Core devices.

Name	Status	Time
<a href="#">CMPE583-GreengrassCore</a>	Healthy	12 minutes 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.

#### Step 1: Select target core devices

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

Target type

Core device

Thing group

Target name

CMPE583-GreengrassCore

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

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

# Step 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 (1) Info**

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

[Find by thing name](#)

Name	Client device associated
CMPE_Test_Client1	

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

< 1 > [View](#)

**Associate client devices with core device**

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

AWS IoT thing name

CMPE\_Test\_Client1 [Add](#) [View AWS IoT things](#)

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

- 1
- 2
- 3

# Step 3: Deploy Required Greengrass Components

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

# Step 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       "MyPermissiveDeviceGroup": {
6         "selectionRule": "thingName: *",
7         "policyName": "MyPermissivePolicy"
8       }
9     },
10    "policies": {
11      "MyPermissivePolicy": {
12        "AllowAll": {
13          "statementDescription": "Allow clients to perform all actions",
14          "operations": [
15            "*"
16          ],
17          "resources": [
18            "*"
19          ]
20        }
21      }
22    }
23  }
```

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

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

# Step 3: Deploy Required Greengrass Components

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

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

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

[Edit configuration](#)

- Relay MQTT messages on the clients/+/hello/world topic filter from client devices to the AWS IoT Core cloud service.
- For example, this topic filter matches the clients/CMPE\_Test\_Client1/hello/world topic.

Configuration to merge

The configuration to merge with the configuration on each core device. The deployment merges this JSON object after it resets the values that you specify in the list of reset paths. [Learn more](#)

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

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

# Step 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

**Deployment successfully created**

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

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

### Deployment for CMPE583-GreengrassCore

Latest revision: 1 ▾ Cancel Actions ▾

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

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

# Step 2: Run Sample Device Discovery

## Verify Client Device Connection From Terminal

```
cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py --thing_name CMPE_Test_Client1 --topic 'clients/CMPE_Test_Client1/hello/world' --message 'Hello World!' --ca_file /home/cagatay/shared_folder/AmazonRootCA1.pem --cert /home/cagatay/shared_folder/certificate.pem.crt --key /home/cagatay/shared_folder/private.pem.key --region eu-central-1 --verbosity Warn
Performing greengrass discovery...
[WARN] [2023-11-04T15:46:21Z] [00007f5a87ffff640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice-CMPE583-GreengrassCore', cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address='10.0.2.15', metadata='', port=8883)]], certificateAuthorities=['-----BEGIN CERTIFICATE-----\nMIIDTCAR2gAwIBAgIVAMDxNyNBew2Jja8e093NSaPTIV8WMA0GCSqGSIB3DQE\n/nCwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBjbmMuMRww\n/nGgYDVQQLDBNbWF6b24gV2Vi\nIFNlcnZpY2VzMRMwEQYDVQQIDApXYXNoaW5ndG9u\n/nMRAwDgYDVQQHDAdTZWFOdGxLMRswGQYDVQQDBJHcmVlbmdyYXNzIENvcmlUgQ0Ew\n/nHhcNMjMxMTA0MTMwMTM5WhcNMjgxMTAyMTMwMTM5WjCBiT\nELMAkGA1UEBhMCVVMx\n/nGDAWBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMB0GA1UECw\nTQW1hem9uIFdlyiBT\n/nZXJ2aWNlc\nzETMBEGA1UECAwKV2FzaGluz3RvbjEQMA4GA1UEBwwHU2VhdHRsZTEb\n/nMB\nkGA1UEAw\nSR3JlZW5ncmFzcyBDb3JlIENB\nMIIBIjANBgkqhkiG9w0BAQEFAAO\n/nAQ8AMII\nB\nCgKCAQEAmZ2lfIUSehjcUmSgkm1u\nCt8VoKcQtWIoy0pzVmyNaR2RnDf\n/n48hu\nzRAT+Ee6dVxyLMRcs\nw1n\ncJM62cPv8wATNDgC/IMdtotLLqwAGKdcFFiWvt\n/nEz1zfN8JsQJ8Mutr4X+PffmFSq\nxSTdfTXB2l7Z0SX0SE7auaLzHcH8HTDYAk\nlp2h\n/nXeK4VMxiKbHne4Kd0PbQgk8m83Bn\niAOWOKOTab9wFjwi+1\nwd875jV3r1SS64uz+9\\ncJ6Pp+4Kv+c\nvjZyMWPTnSKI4QoXlbQd+92lXzwqtGCTvzAPdLm5E/B8ber2/lb2s\\nwJ4f8A7jeQs5VtDz\nmVoeXLd8Sdm/ocji4+pGqQIDAQABozIwMDAPB\ngNVHRMBAf8E\\nBT\nADAQH/MB0GA1UdDgQWBBQyiqUi\npUY0XyyDknCF/imtYlYyLzANBgkqhkiG9w0B\\nAQSFAA0CAQEAYfoiiBX5fNJRpm/MW1VVdE62f7K+30atTPoroZa6CTgULZhdTnWG\\nWP5MIZt9iKq3AQzgma7aQEt/dKJJL3nmDqsY/fHRjjjcF6SSu1W4Fp32BFp3m57\\naeICvFqMgHztyBB5WJVsU7Ek\nlWSn0PjWSgcBWticXFw8a0G55HS5rrUqj9mBYWSm\\njo/7rVsX/U0kUHP19sBu6JrJarvibkUdyxY66m0EpLMmf9My/11TIL5s2rhpqRSX\\nR9DsiyGAq7KXaJzMsxWn1DxvlyRfjNo3R3W0ICWq1F6y6RtwU+ofAoTxMbC25XQ0\\ncYI/qnKkj0/K4ycY0QPZN7qr76I5INP5iA==\\n-----END CERTIFICATE-----\\n'])
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/hello/world: {"message": "Hello World!", "sequence": 0}

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

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

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

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

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

## Verify From MQTT Test Client

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

**Subscribe to a topic:**

- Topic filter:** clients/+hello/world
- Info:** The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.
- Additional configuration:** A link to configuration options.
- Subscribe:** A button to subscribe to the topic.

**Publish to a topic:** This section is currently empty.

**Subscriptions:** clients/+hello/world

- Subscriptions:** clients/+hello/world (with a heart icon and an X)
- Message history:** Three messages from clients/CMPE\_Test\_Client1/hello/world, all published on November 04, 2023, at 18:46:25 (UTC+0300).

A large orange arrow points to the "Topic filter" input field in the "Subscribe to a topic" section. A dashed orange circle highlights the message history area.

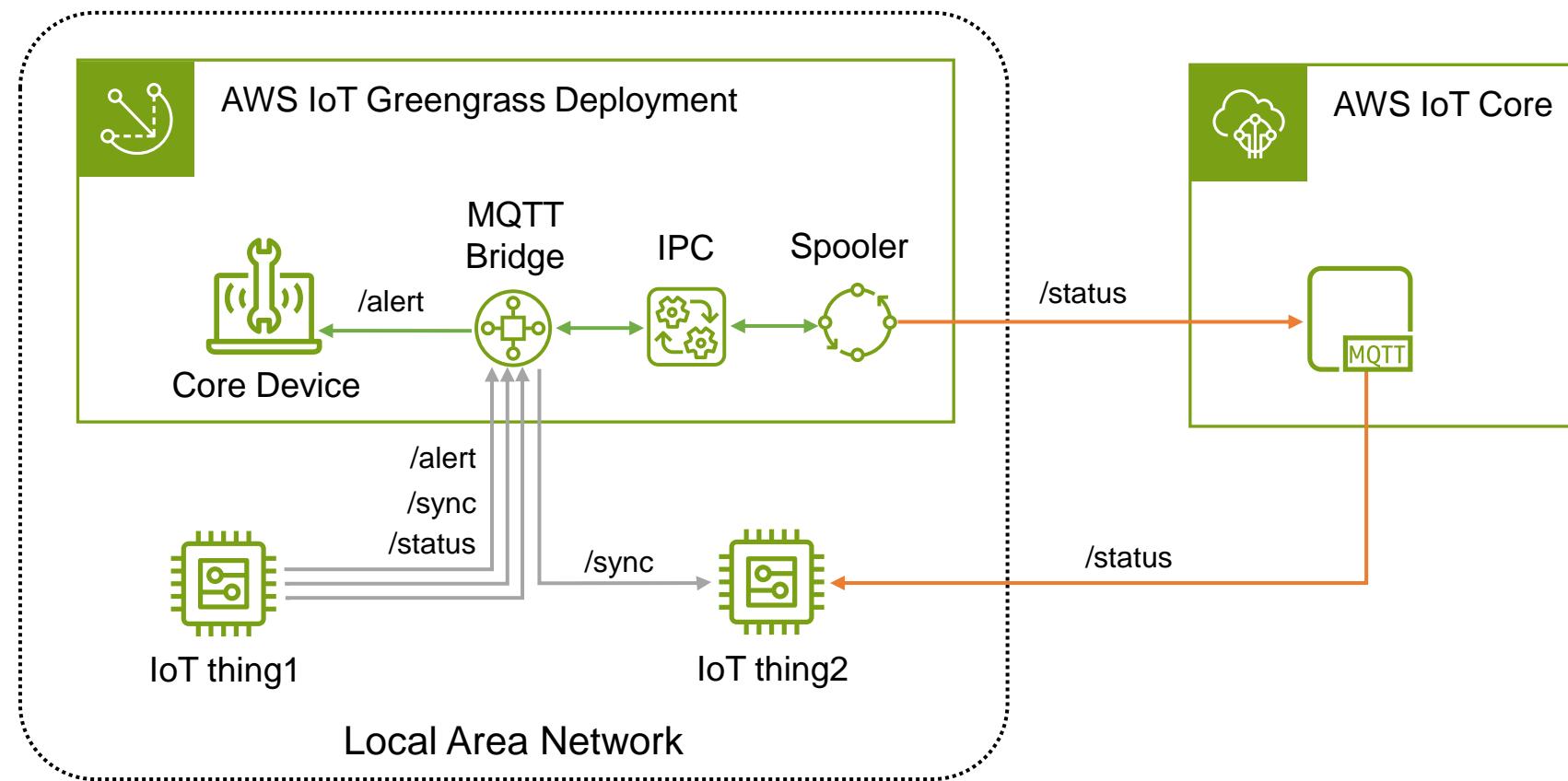
Topic	Time
clients/CMPE_Test_Client1/hello/world	November 04, 2023, 18:46:25 (UTC+0300)
clients/CMPE_Test_Client1/hello/world	November 04, 2023, 18:46:25 (UTC+0300)
clients/CMPE_Test_Client1/hello/world	November 04, 2023, 18:46:24 (UTC+0300)

# Communicate with Client Devices

---

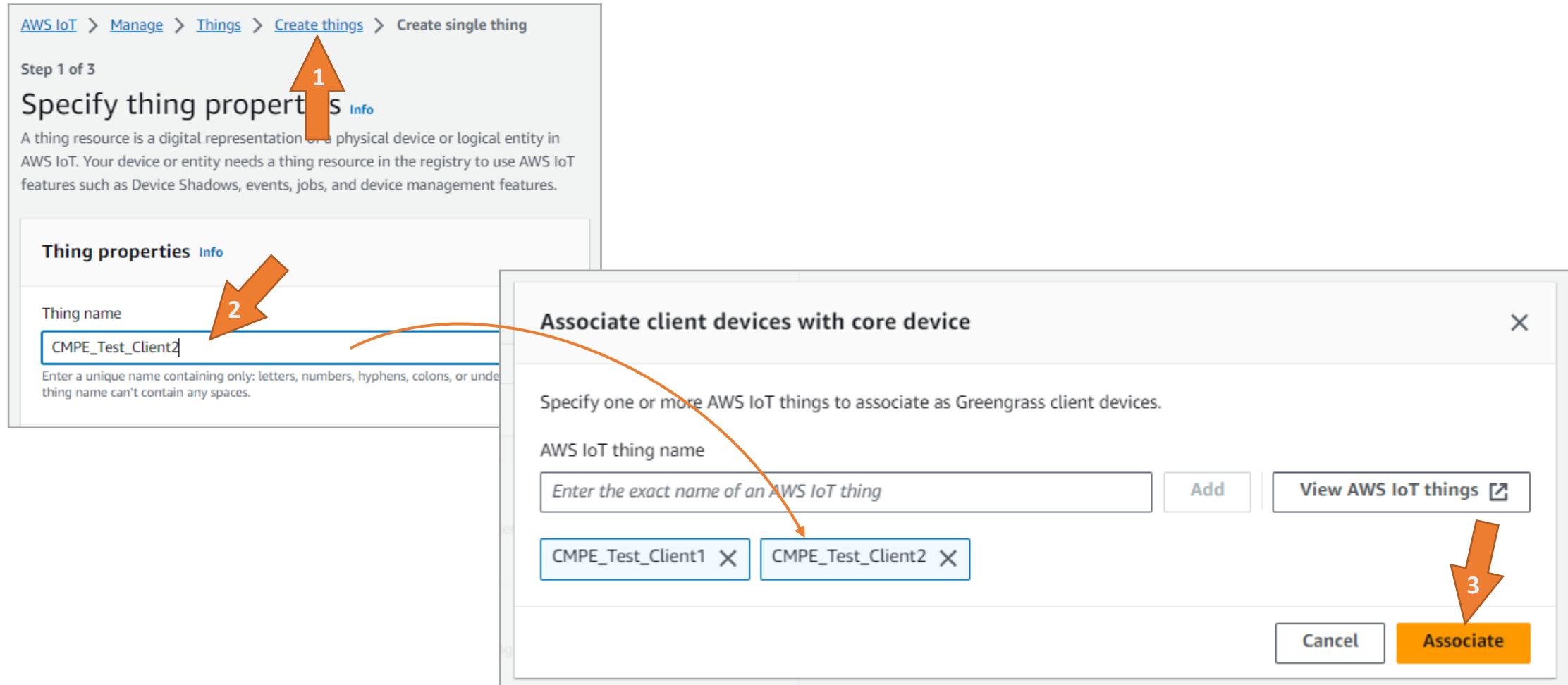
# Communicate with Client Devices

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

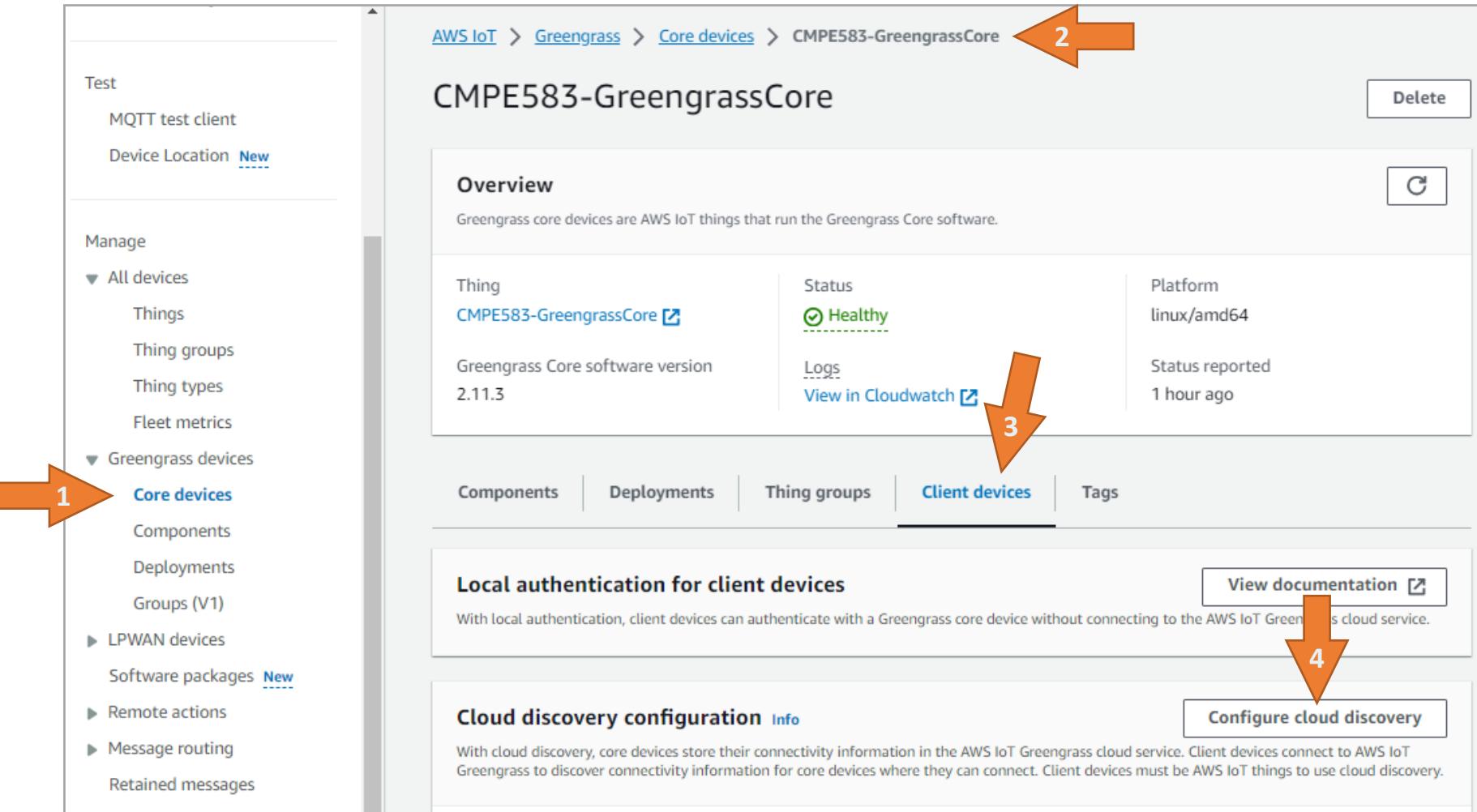


# Step 1: Create More IoT Clients

Create Another Client and Associate It With Core Device



# Step 2: Reconfigure Cloud Discovery



# Step 2: Reconfigure Cloud Discovery

## Reconfigure MQTT Bridge

**Step 3: Configure and deploy Greengrass components**

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

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

**Edit configuration**



**Configuration to merge**

The configuration to merge with the configuration on each core device. The deployment merges this JSON object after it resets the values that you specify in the list of reset paths. [Learn more](#)

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

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

# Step 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](#)

```
8▼  "ClientToClientEvents": {  
9    "topic": "clients/+sync",  
10   "source": "LocalMqtt",  
11   "target": "Pubsub"  
12 },  
13▼  "ClientDeviceEvents": {  
14    "topic": "clients/+alert",  
15    "targetTopicPrefix": "event/",  
16    "source": "LocalMqtt",  
17    "target": "Pubsub"  
18 },  
19▼  "ClientDeviceCloudStatusUpdate": {  
20    "topic": "clients/+status",  
21    "targetTopicPrefix": "update/",  
22    "source": "LocalMqtt",  
23    "target": "IotCore"  
24 }  
25 }  
  
JSON  Ln 26, Col 2  ✖ Errors: 0  ⚠ Warnings: 0  ⚙
```

Relay messages from client devices to Greengrass publish/subscribe message broker through MQTT Bridge on Core Device with **clients/+sync** topic filter.

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

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

# Step 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 successfully created**

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

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

### Deployment for CMPE583-GreengrassCore

Latest revision: 1 ▾ Cancel Actions ▾

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

# Step 3: Create a Custom Component

## To Suscribe Local MQTT Events

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

# Step 3: Create a Custom Component

## Configure accessControl for Local Messages

The screenshot shows the AWS Greengrass Component Source configuration interface. On the left, there's a radio button group for "Component source": "Enter recipe as JSON" (selected), "Enter recipe as YAML", and "Import Lambda function". Below this is a "Recipe" section with a note about artifact availability in an S3 bucket. To the right, a large JSON code editor displays the component's recipe. An orange arrow points from the "Enter recipe as JSON" label to the "Enter recipe as JSON" radio button. The JSON code is as follows:

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

Annotations on the right side explain specific parts of the JSON:

- A callout box points to the "Enter recipe as JSON" section with the text: "Check GitHub to see whole contents in **cmpe583-ClientDeviceEventSubscriber.json**".
- An annotation points to the "aws.greengrass.ipc.pubsub" section with the text: "Allow the component to subscribe to messages from local IoT Devices."
- An annotation points to the "resources": ["\*"] section with the text: "Allow access to all topics."

# Step 3: Create a Custom Component

## Configure Artifacts in the Recipe

Component source

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

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

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

Recipe

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

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

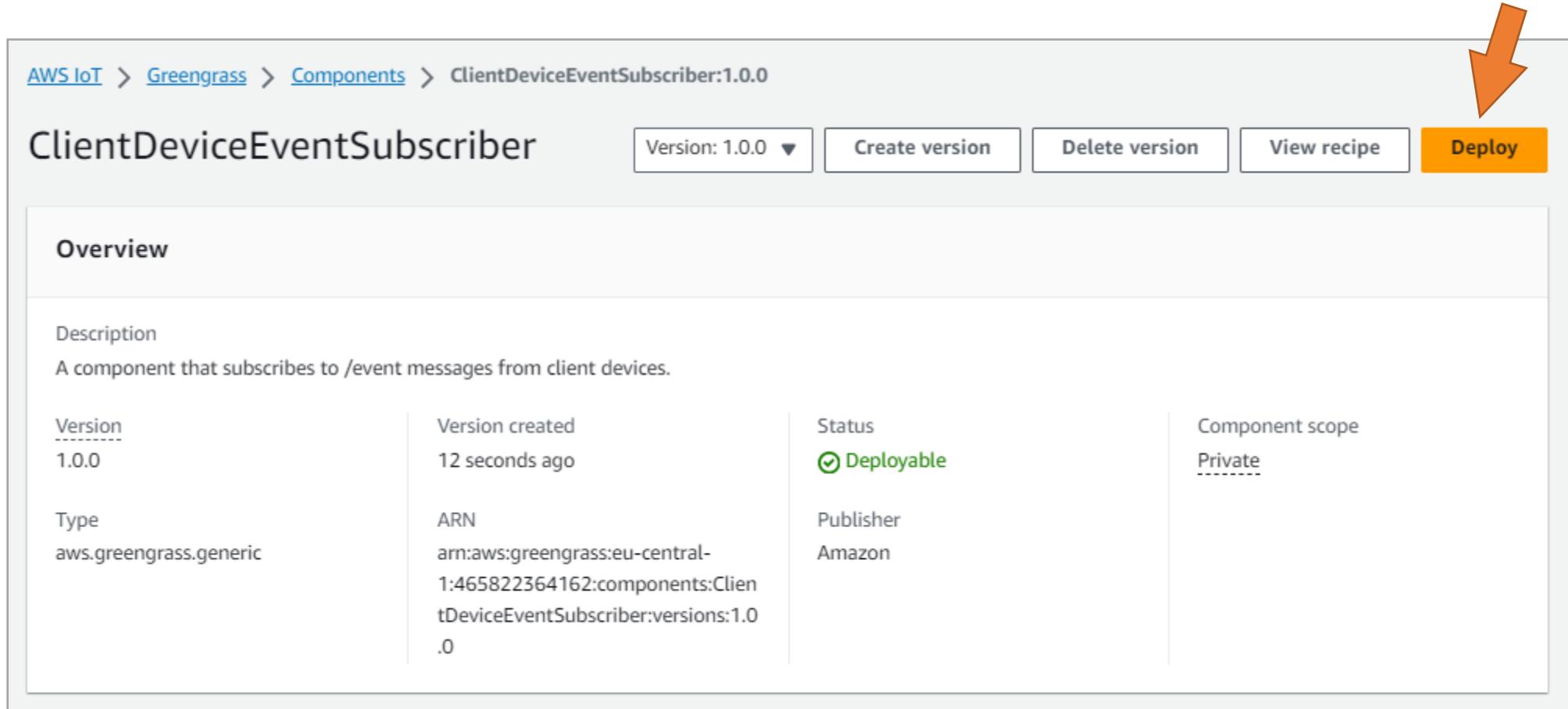
JSON Ln 53, Col 2 Errors: 0 Warnings: 0

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

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

# Step 4: Deploy Your Component

## Via AWS IoT Greengrass (Console)



AWS IoT > Greengrass > Components > ClientDeviceEventSubscriber:1.0.0

ClientDeviceEventSubscriber

Version: 1.0.0 ▾ Create version Delete version View recipe Deploy

**Overview**

Description

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

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

# Step 4: Deploy Your Component

## Select the Component

**Select components - optional**

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

**My components (1)**

<input type="checkbox"/>	Name 
<input checked="" type="checkbox"/>	<a href="#">ClientDeviceEventSubscriber</a>

**Public components (47)**

<input type="checkbox"/>	Name 
<input checked="" type="checkbox"/>	<a href="#">aws.greengrass.Nucleus</a>
<input checked="" type="checkbox"/>	<a href="#">aws.greengrass.clientdevices.Auth</a>
<input checked="" type="checkbox"/>	<a href="#">aws.greengrass.clientdevices.IPDetector</a>
<input checked="" type="checkbox"/>	<a href="#">aws.greengrass.clientdevices.mqtt.Bridge</a>
<input checked="" type="checkbox"/>	<a href="#">aws.greengrass.clientdevices.mqtt.Moquette</a>



# 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 second

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

Failure handling policy  
Rollback

Download as JSON Cancel Previous Deploy



# Step 4: Deploy Your Component

## Check Deployment Status on AWS IoT Console

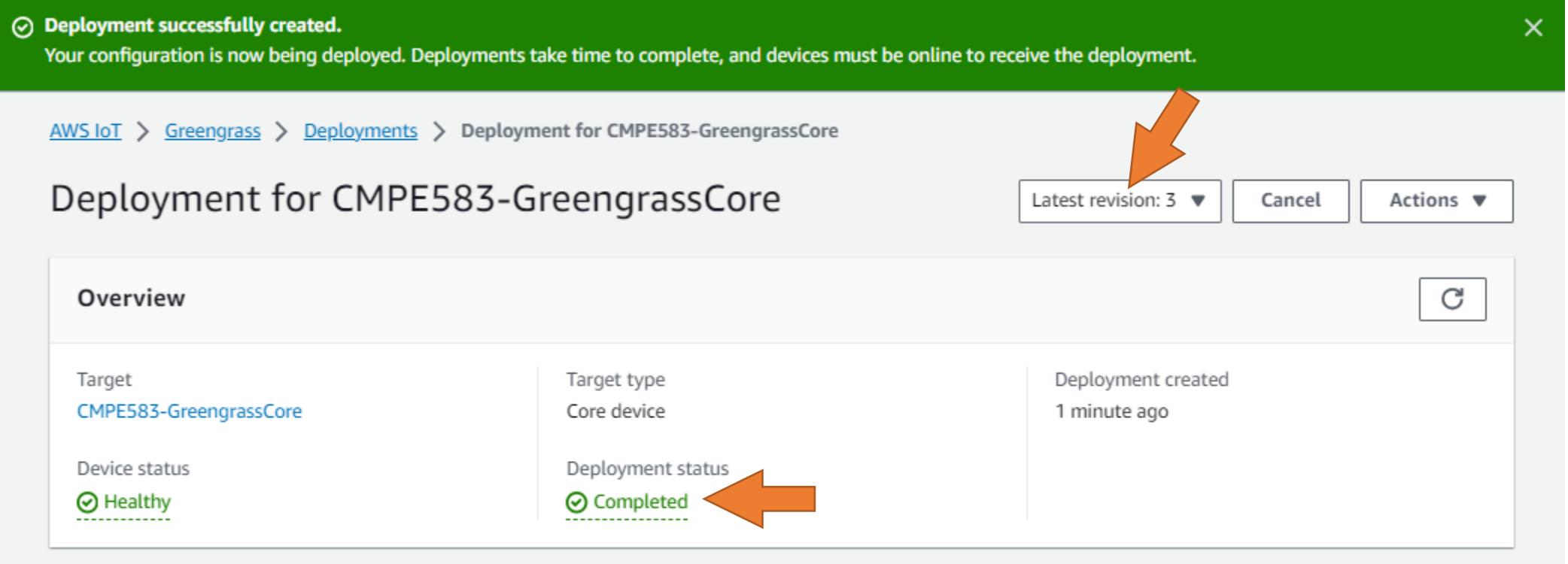
Deployment successfully created.  
Your configuration is now being deployed. Deployments take time to complete, and devices must be online to receive the deployment.

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

### Deployment for CMPE583-GreengrassCore

Latest revision: 3 ▾ Cancel Actions ▾

Overview	C	
Target CMPE583-GreengrassCore	Target type Core device	Deployment created 1 minute ago
Device status <span style="color: green;">健康的</span>	Deployment status <span style="color: green;">完成</span>	



# Step 4: Deploy Your Component

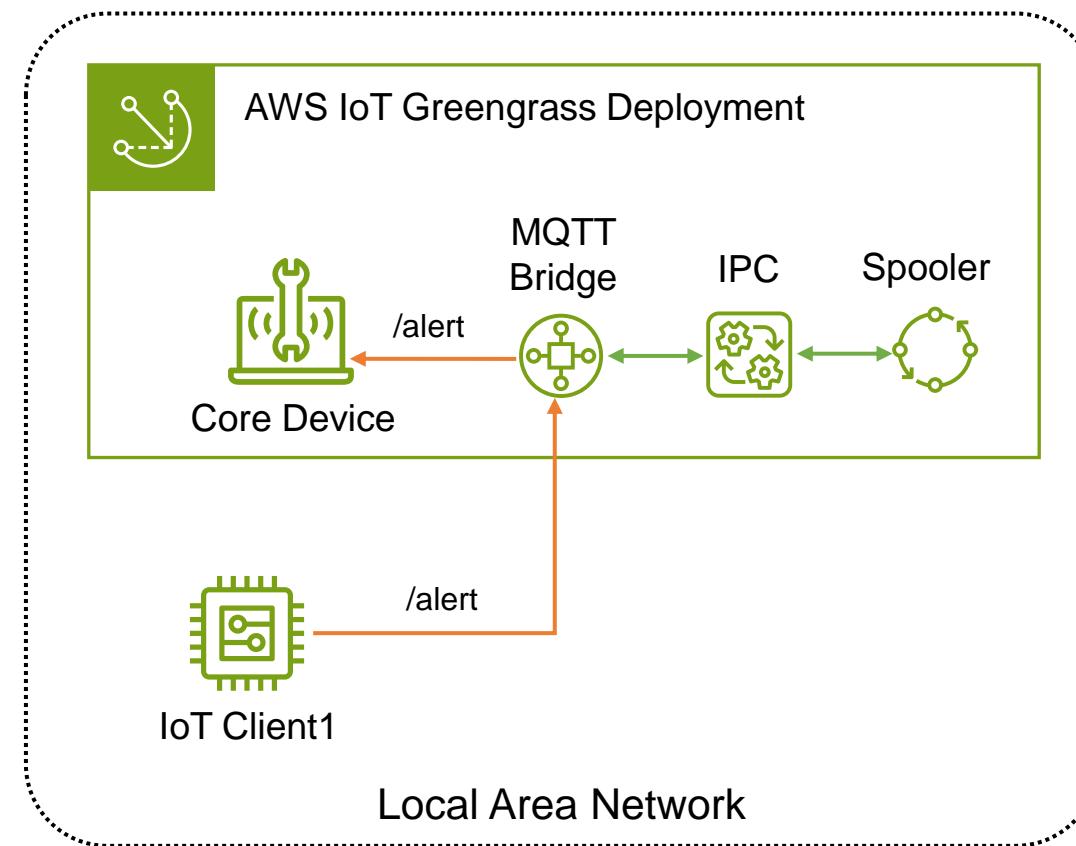
## Check Deployment Status on Core Device

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

# Case 1: IoT Client to Core Device Communication

- Publish 'clients/+ alert' Topic From IoT Client1 to Core Device



# Publish 'clients/+/alert' Topic From IoT Client

## Publish Topic to Trigger Subscribers (Core Device)

- Publish the topic with basic\_discovery.py script.

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

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

This message will be relayed to Core Device

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

# Step 1: Publish 'clients/+/alert' Topic From IoT Client Verify Process on Terminal

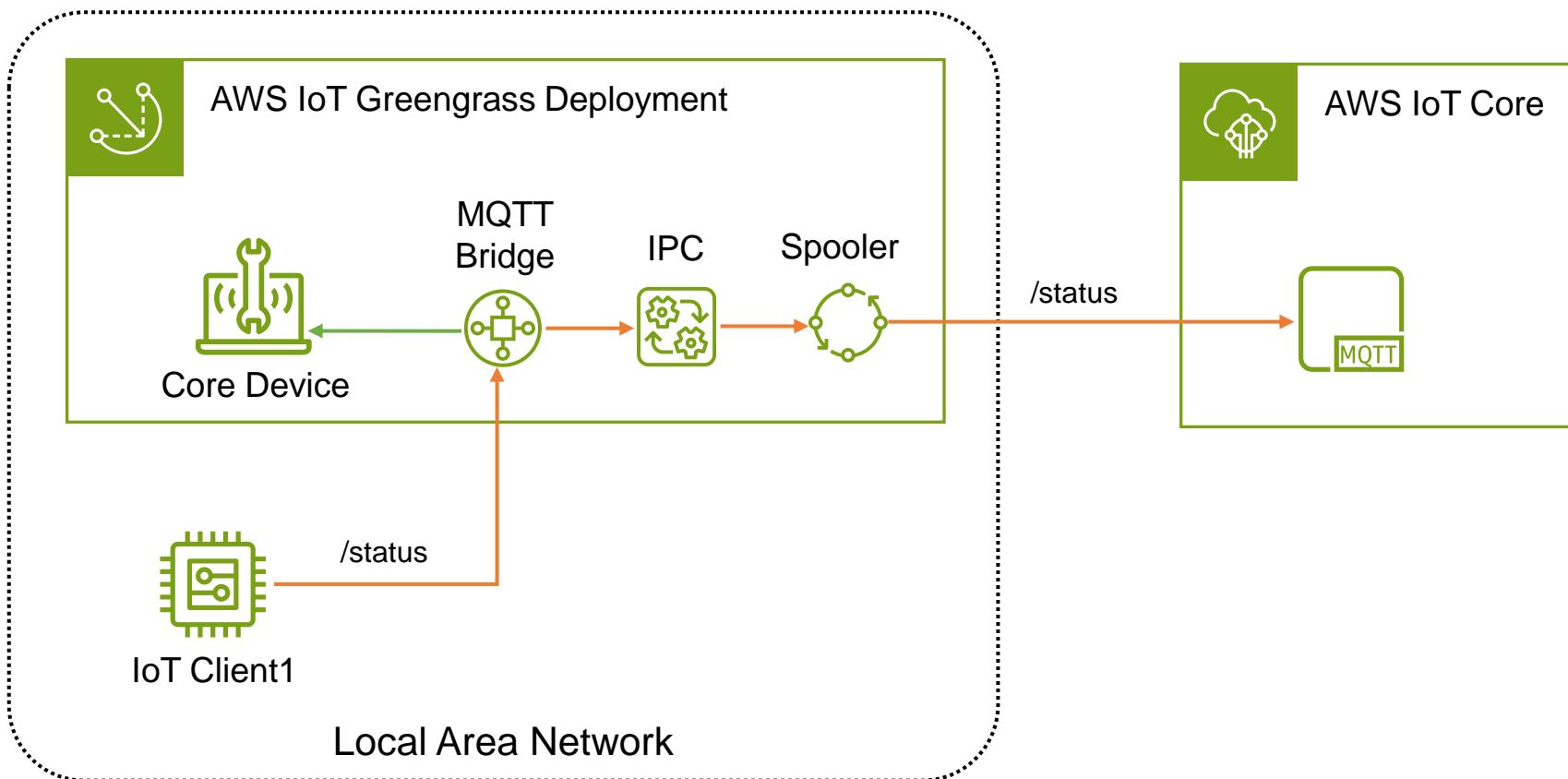
# Step 2: Publish 'clients+/alert' Topic From IoT Client Verify Process on Core Device's Terminal

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

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

# Case 2: IoT Client to IoT Core Communication

- Publish 'clients/+status' Topic From IoT Client1 to AWS IoT Core



# Publish 'clients/+/status' Topic From IoT Client

## Publish Topic to Trigger Subscribers (AWS IoT Core)

- Publish the topic with basic\_discovery.py script.

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

This message will be relayed  
to AWS IoT Core

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

# Step 1: Publish 'clients/+/status' Topic From IoT Client Verify Process on Terminal

```
cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
--thing_name CMPE_Test_Client1 \
--topic 'clients/CMPE_Test_Client1/status' \
--message 'Client device is running for 2 days without error!' \
--ca_file /home/cagatay/shared_folder/AmazonRootCA1.pem \
--cert /home/cagatay/shared_folder/certificate.pem.crt \
--key /home/cagatay/shared_folder/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1
Performing greengrass discovery...
[WARN] [2023-11-05T08:41:03Z] [00007f61829fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice-CMPE583-GreengrassCore', cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address='10.0.2.15', metadata='', port=8883)]), certificateAuthorities=['-----BEGIN CERTIFICATE-----\nMIID1TCCAr2gAwIBAgIVAMDxNyNBeW2Jja8e093NSaPTIV8WMA0GCSqGSIb3DQE\nBnCwUAMIGJMQswCQYDVQ\nQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBjbmMuMRww\\nGgYDVQQLDBNbWF6b24gV2ViIFNlc\nzP\nY2VzMRMwEQYDVQQIDApXYXNoaW5ndG9u\\nMRAwDgYDVQQHAdTZWF0dG\nxLMRswGQYDVQQDDBJHcmVlbmdyYXNzIENvc\nmUgQ0Ew\\nHhcNMjMxMTA0MTMwMTM5WhcNMjgxMTAyMTMwMTM5WjCBiTELMAkGA1UEBhMCVVMx\\nGDAwBgNVBAoMD0FtYXpvbi5jb2\n0gSW5jLjEcMBoGA1UECwwTQW1hem9uIFd1YiBT\\nZXJ2aWNlc\nzETMBEGA1UECAwKV2FzaGluZ3RvbjEQMA4GA1UEBwwHU2VhdHRsZTEb\\nMBkGA1UEAw\nSR3JlZW5ncmFzcyBD\nb3JLIENB\nMIIBjANB\nbgkqhkiG9w0BAQEFAAO\nC\\nAQ8AMII\nBCgKCAQEAmZ2lfIUSehjcUmSgkm1uCt8VoKcQtWIoy0pzVmyNaR2RnDFr\\n48huzRAT+Ee6dVxyLMRcs\nw1ncJM62cPv8w\nATNDgC/IMdtotLLqwAGKdcFfFiWvtx\\nE\nZ1zfN8jsQJ8Mutr4X+PfFmFSqxSTdfTXB217Z0SX0SE7auaLzHcH8HTDYAklp2h\\nXeK4VMxiKbHne4KdoPbQgk8m83BniAO\nWOK0Tab\n9wFjwi+1wd875jV3r1SS64uz+9\\ncJ6Pp+4Kv+cvjZyMWPTnSKI4QoXlbQd+92lXZwqtGCTvzAPdLm5E/B8ber2/lb2s\\nwJ4f8A7jeQs5VtDz\nmVoeXLd8Sdm/ocji4+pGqQIDAQ\nABozIwMDAPBgnVHRMBAf8E\\nBTADAQH/MB0GA1UdDgQWBBQyiqUi\npUYO\nXyyDknCF/imtYlYyLzANB\nbgkqhkiG9w0B\\nA\nQsFAA0CAQEA\nYfoiiBX5fNJRpm/MW1VVdE62f7K+30atTP\noroZa6CTgULZhdTnWG\\nWP5MIZt9iKq3A\nQzgma7aQEt/dKJJL3nmDqsY/fH\nRjjjcF6SSu1W4Fp32BFp3mi57\\naeICvFqMgHztyBB5WJVsU7EktWSn0PjWSgcB\nWticXFw8a0G55H\nS5rrUqj9mBYWSm\\njo/7rVsX/U0kUHP19sBu6JrJarvibkUdyxY66m0EpLMmf9My/11TIL5s2rh\npqRSX\\nR9DsiyGAq7KXaJzMsxWn1DxvlyRfjNo3R3W0ICWq1F6y6RtwU+ofAo\nTxMbC25XQ0\\ncYI/qnKkj0/K4ycY0QPZN7qr76I5INP5iA==\\n----END CERTIFICATE----\\n')])]\nTrying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883\nConnected!\nPublished topic clients/CMPE_Test_Client1/status: {"message": "Client device is running for 2 days without error!", "sequence": 0}\n\nPublish received on topic clients/CMPE_Test_Client1/status\nb'{"message": "Client device is running for 2 days without error!", "sequence": 0}'
```

# Step 2: Publish 'clients+/status' Topic From IoT Client

## Verify Process on MQTT Test Client

The screenshot shows an MQTT test client interface with two main sections: 'Subscribe to a topic' and 'Publish to a topic'. In the 'Subscribe to a topic' section, there is a 'Topic filter' input field containing 'update/clients+/status'. An orange arrow points to this input field. Below it is a 'Subscribe' button. In the 'Subscriptions' section, there is a list of subscriptions with 'update/clients+/status' selected. Underneath, a detailed view of a received message is shown, with the message content highlighted by a red dotted box.

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

**Topic filter** | [Info](#)  
The topic filter describes the topics which you want to subscribe. The topic filter can include MQTT wildcard characters.

update/clients+/status

► Additional configuration

**Subscribe**

**Subscriptions**

update/clients+/status

Pause   Clear   Export   Edit

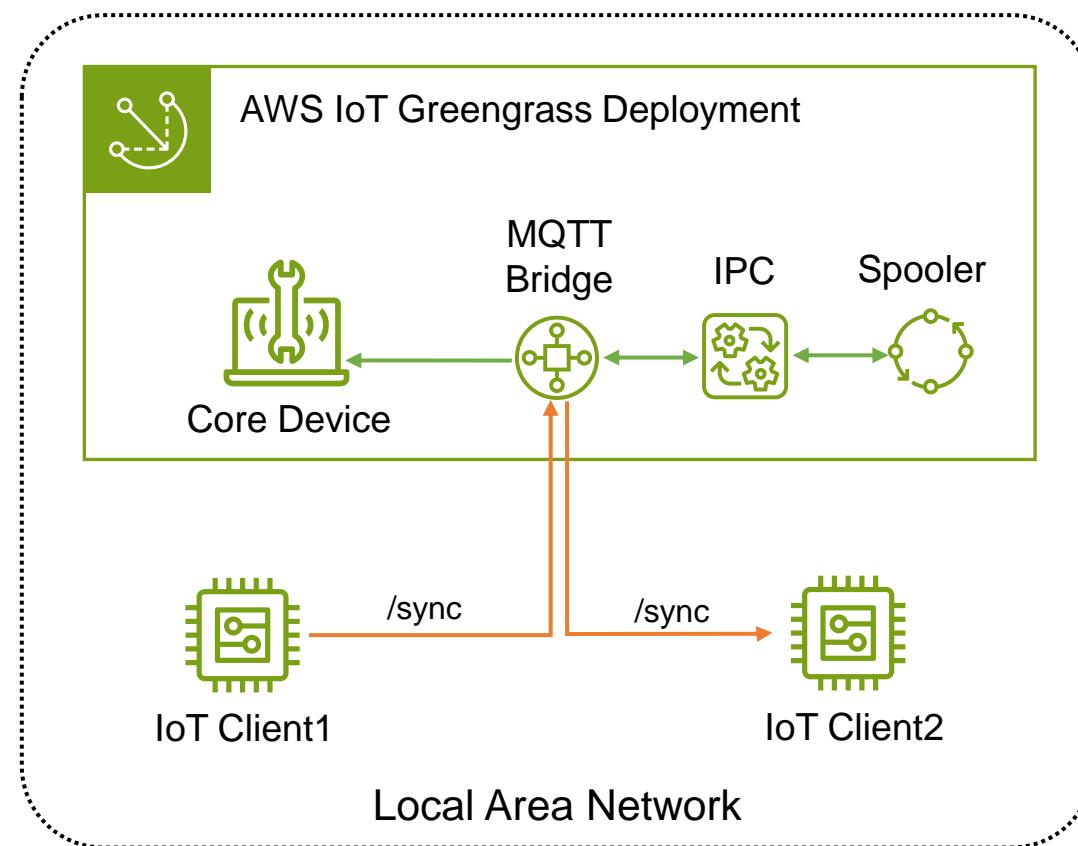
update/clients+/status

▼ update/clients/CMPE\_Test\_Client1/status      November 05, 2023, 11:43:37 (UTC+0300)

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

# Case 3: IoT Client to IoT Client Communication

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



# Publish 'clients/+/sync' Topic From IoT Client Subscribe Topic From Test Client 2

- Subscribe the topic using the '*subscribe*' --mode argument of basic\_discovery.py script.

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

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

# Publish 'clients/+/sync' Topic From IoT Client

## Publish Topic to Trigger Subscribers (Test Client 2)

- Publish the topic using the ‘publish’ --mode argument of basic\_discovery.py script.

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

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

This message will be relayed to test client 2 through local Greengrass core MQTT broker

★ The MQTT bridge configuration is only applicable when routing messages between the Greengrass core and AWS IoT Core. Things-to-things messages are routed through the local Greengrass core MQTT broker.

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

## Verify Topic Published

```
(venvclient1) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
--thing_name 'CMPE_Test_Client1' \
--topic 'clients/CMPE_Test_Client1/sync' \
--message 'Hello from IoT Client 1!' \
--ca_file ~/shared_folder/AmazonRootCA1.pem \
--cert ~/shared_folder/certificate.pem.crt \
--key ~/shared_folder/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1 \
--mode publish
Performing greengrass discovery...
[WARN] [2023-11-06T13:52:20Z] [00007fd3ac8fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice22364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address='AwIBAgIVAJSS5wdMBGrA7mIHUCs183HgzuqAnMA0GCSqGSIb3DQEB\ncwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBjbmMuMRww\nGgYDVQDNzIENvcnUgQ0Ew\nnHcNMjMxMTA2MTE1ODE0WhcNMjgxMTA0MTE1ODE0WjCBiTELMAkGA1UEBhMCVVMx\nnGDAWBgNVBAoMD0FtYXpbvi5jb20gSW5jLjEcMB0GA1UE0FzcyBDb3JlIENBmIIBIjANBgkqhkiG9w0BAQEFAAOAcnAQ8AMIIIBCgKCAQEAsfbYoDMM56kG2MLGqo3/6RUFDquIN97/4qQ/CD727F45WHPd\nnuwbBtACsSXXSvc7F3k1Z3\nnwhEobHrDR8hFmjBQ5y1u0Wj3mIpScCELAtudjScHHQV0dbcium+f0vn+u0hIV0x0\nnYydkscLyVyPYcL0h7D7B0n1hsUv3WT0gYcmh1zSN49/29D9UdG07i01rWQrz9sINS7QZLNS1zANBgkqhkiG9w0B\nnAQSFAAOCAQEAMd5Pmw42K0ZWvV9ZAM1W3g2D4egNa98jBogP8otc4d0g3vhtFtVF\nn+jWWF5kinHsMVv0uQTjxV6Rf17PiYCRts3xobU47Xo5w2nl4Z+IuCJGeT3Lkm1arGR4is4c2SpImAdUkjEU8SGh\nnHEkifLNFejNBx/wurqPyPRk8yw6R9c1j7ExHFHlYRLpzBguUfcy+pNDUiP E6A4jHpv
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/sync: {"message": "Hello from IoT Client 1!", "sequence": 0}
```

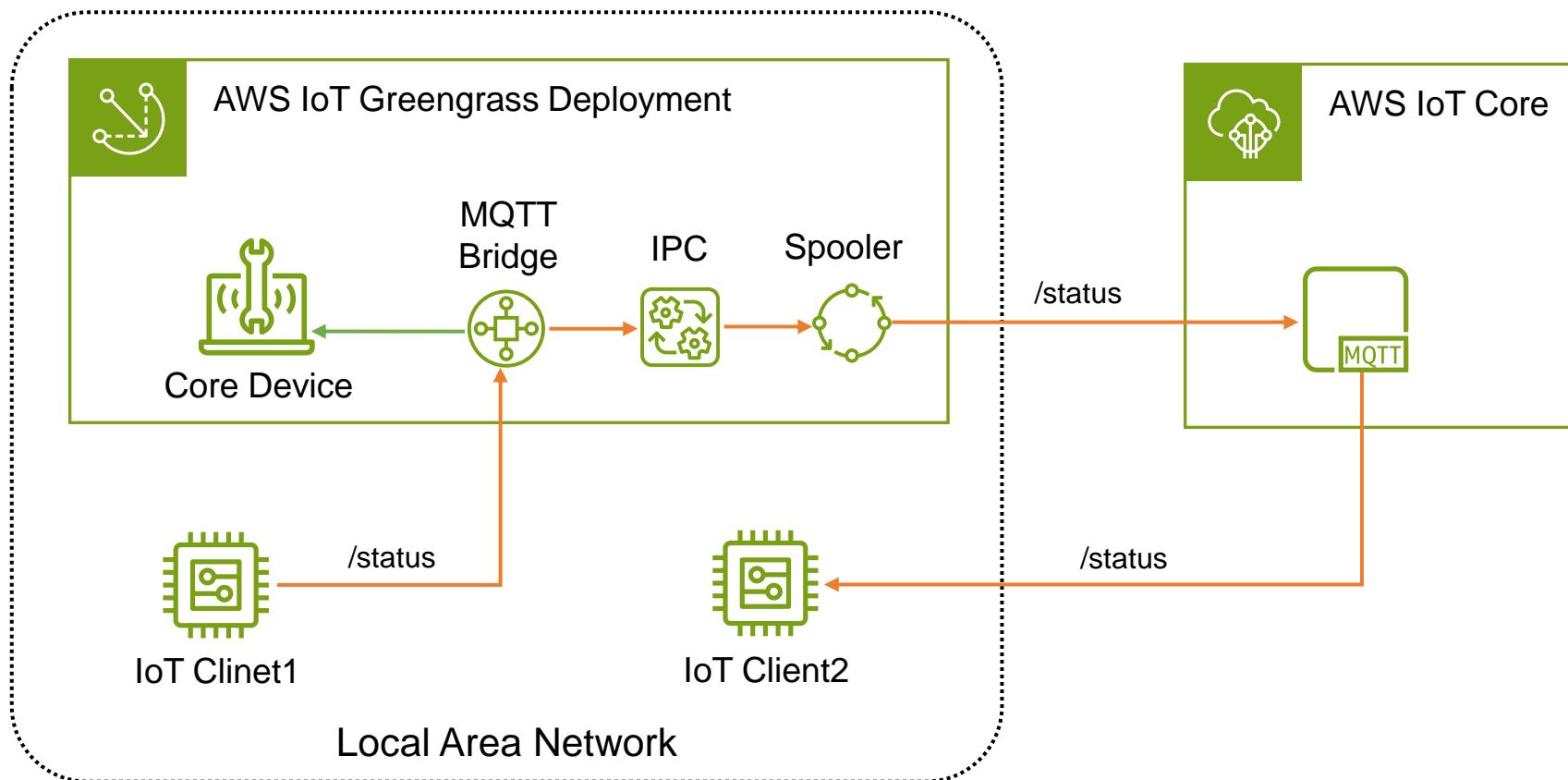
# Step 2: Publish 'clients/+/sync' Topic From IoT Client

## Verify Topic Received

```
(venvclient2) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
--thing_name 'CMPE_Test_Client2' \
--topic 'clients/+/sync' \
--ca_file ~/shared_folder/AmazonRootCA1.pem \
--cert ~/shared_folder/client2-certificate.pem.crt \
--key ~/shared_folder/client2-private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1 \
--mode subscribe
Performing greengrass discovery...
[WARN] [2023-11-06T13:51:22Z] [00007fd0277fe640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDev
22364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_addre
AwIBAgIVAJ5S5wdMBGrA7mIHUCs183HgzuqAnMA0GCSqGSIb3DQEByCwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBjbmMuMRww\nGgYD
NzIEvcmUgQ0Ew\nHhcNMjMxMTA2MTE1ODE0WhcNMjgxMTA0MTE1ODE0WjCBiTELMAkGA1UEBhMCVVMx\nGDAwBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMBoGA1U
FzcyBDb3JlIENBMIIBjANBgkqhkiG9w0BAQEFAAOC\nAQ8AMIIIBCgKCAQEAsfbYoDMM56kG2MLGqo3/6RUFDquIN97/4qQ/CD727F45WHPd\nnuwbBtACsSXXSvc
k1Z3\nwhEobHrDR8hFmjBQ5y1u0Wj3mIpScCELAtudjSChHQV0dbcium+f0vn+u0hIV0x0\nYydkscLyVyPYcL0h7D7B0n1hsUv3WT0gYcmh1zSN49/29D9UdG07
rWQrz9sINST7QZLNS1zANBgkqhkiG9w0B\nAQsFAAACQEAQEMd5Pmw42K0ZWvV9ZAM1W3g2D4egNa98jBogP8otc4d0g3vhtFtVF\n+jWWF5kinHsMVvOuQTjxV6Rf
iYCRts3xobU47Xo5w2nl4Z+IuCJGeT3Lkm1arGR4is4c2SpImAdUkjEU8SGh\nHEkifLNFejNBx/wurqPyPRk8yw6R9c1j7ExHFhLYRLpzBgfUfcy+pNDUi
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

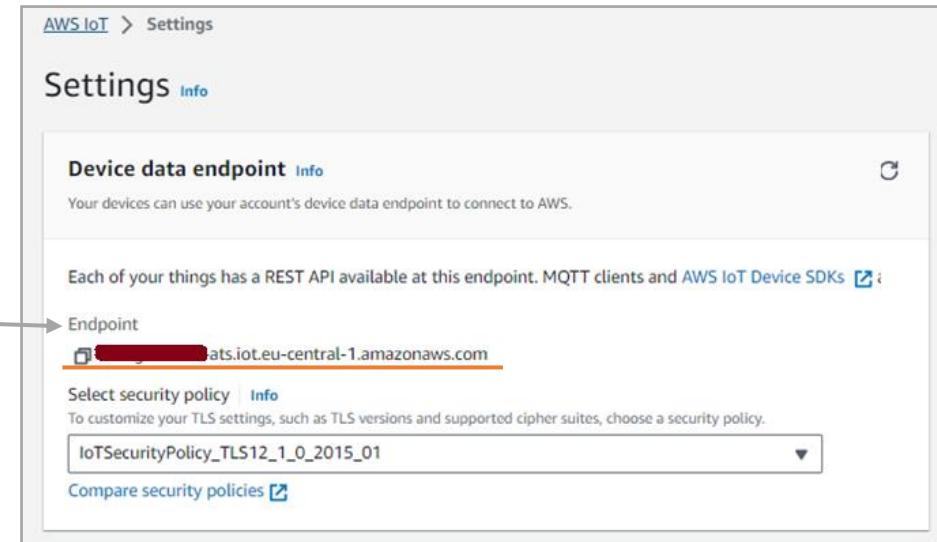
- Publish 'clients/+status' Topic From IoT Client1 to IoT Client2 through IoT Core



# Receiving Events From AWS IoT Core

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

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



# Publish 'clients/+/update' Topic From IoT Client Subscribe Topic From Test Client 2

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

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

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

# Publish 'clients/+/update' Topic From IoT Client

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

- Publish the topic using the '*publish*' --mode argument of basic\_discovery.py script.

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

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

This message will be relayed  
to AWS IoT Core

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

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

## Verify Topic Published

```
(venvclient1) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
--thing_name 'CMPE_Test_Client1' \
--topic 'clients/CMPE_Test_Client1/status' \
--message 'Status update from Client 1!' \
--ca_file ~/shared_folder/AmazonRootCA1.pem \
--cert ~/shared_folder/certificate.pem.crt \
--key ~/shared_folder/private.pem.key \
--region eu-central-1 \
--verbosity Warn \
--max_pub_ops 1 \
--mode publish
Performing greengrass discovery...
[WARN] [2023-11-06T15:00:26Z] [00007f511e1fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDev
22364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_
addr='AwIBAgIVAJ55wdMBGrA7mIHUCs183HgzuqAnMA0GCSqGSIb3DQEBy\nCwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBjbmMuMRww\nGyD
NzIEvcmUgQ0Ew\nHhcNMjMxMTA2MTE1ODE0WhcNMjgxMTA0MTE1ODE0WjCBiTELMAkGA1UEBhMCVVMx\nGDAWBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMB0GA1U
FzcyBDb3JlIENBMMIBIjANBgkqhkiG9w0BAQEFAAOCAQKCAQEAsfbYoDMM56kG2MLGqo3/6RUFdquIN97/4qQ/CD727F45WHPd\nnuwbBtACsSXXSv
k1Z3\nwhEobHrDR8hFmjBQ5y1u0Wj3mIpScCELAtudjScHHQV0dbcium+f0vn+u0hIV0x0\nYydkscLyVpYcL0h7D7B0n1hsUv3WT0gYcmh1zSN49/29D9UdG07i
rWQrz9sINS7QZLNS1zANBgkqhkiG9w0B\nAQsFAA0CAQEAMd5Pmw42K0ZWvV9ZAM1W3g2D4egNa98jBogP8otc4d0g3vhtFtVF\n+jWWF5kinHsMVv0uQTjxV6Rf1
iYCRts3xobU47Xo5w2nl4Z+IuCJGeT3LKm1arGR4is4c2SpImAdUkjEU8SGh\nHEkifLNFejNBx/wurqPyPRk8yw6R9c1j7ExHFHlYRLpzBgUfcy+pNDUi
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/status: {"message": "Status update from Client 1!", "sequence": 0}
```

# Step 2: Publish 'clients/+/sync' Topic From IoT Client

## Verify Topic Received

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

# Thank You

