Edge Computing with AWS IoT Greengrass - *Part #1*

by Cagatay Sonmez, 31 Nov 2025

The Primary Sources Used in This Lecture

- https://docs.aws.amazon.com/greengrass/v2/developerguide
- https://docs.aws.amazon.com/greengrass/v2/developerguide/setting-up.html
- https://docs.aws.amazon.com/greengrass/v2/developerguide/develop-greengrass-components.html
- https://docs.aws.amazon.com/greengrass/v2/developerguide/ip-detector-component.html
- https://docs.aws.amazon.com/greengrass/v2/developerguide/interprocess-communication.html
- https://docs.aws.amazon.com/greengrass/v2/developerguide/run-lambda-functions.html
- https://docs.aws.amazon.com/greengrass/v2/developerguide/interact-with-local-iot-devices.html
- https://docs.aws.amazon.com/greengrass/v2/developerguide/client-devices-tutorial.html

- All the source codes used in this Lecture can be found in the following GitHub repository:
  - https://github.com/CagataySonmez/AWS-IoT-Greengrass-Demo

Prerequisite: Amazon Web Services (AWS)

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

Our Core Services in this Lecture (Identity & Storage)

IAM (Identity & Access Management)

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

S3 (Simple Storage Service)

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

Our Core Services in this Lecture (Compute & Messaging)

AWS Lambda

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

AWS IoT Core

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

Key Concept to Remember:

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

⚠ Reminder

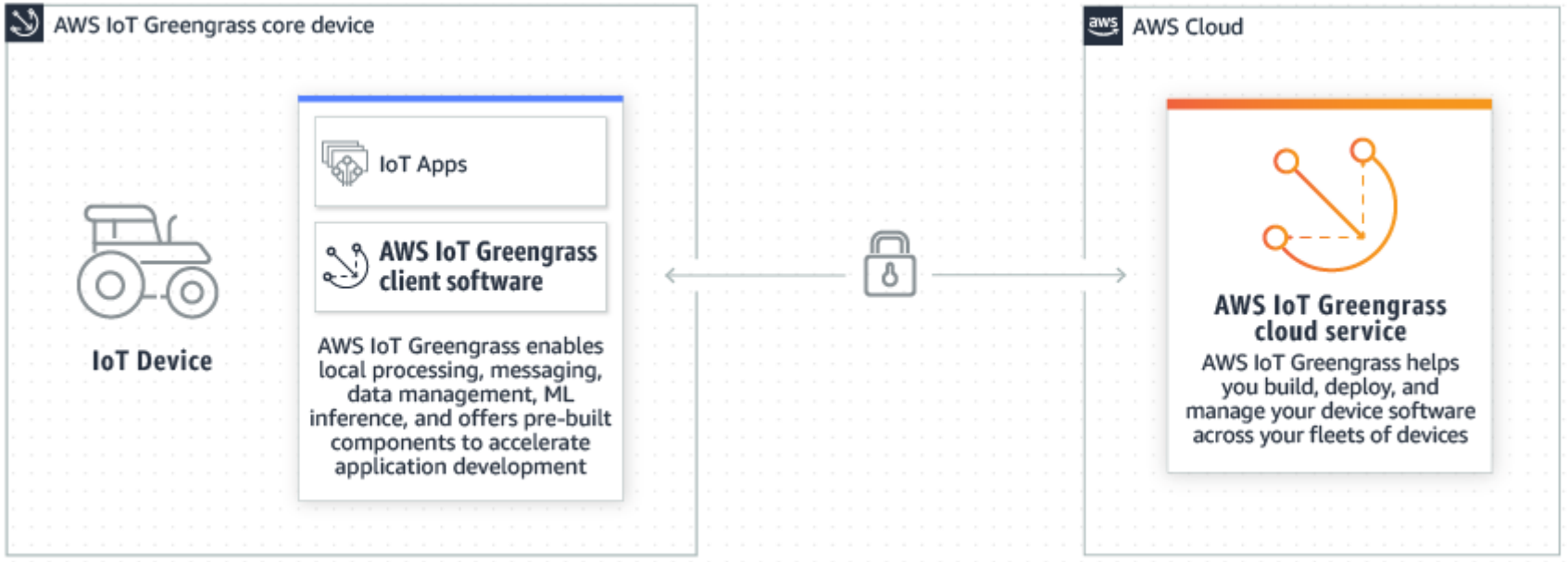
Monitor AWS **Free Tier** usage via **Budget Alerts** to avoid unexpected charges!

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

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

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[1].



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

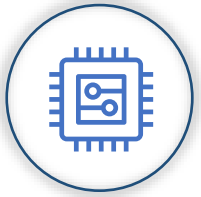
AWS IoT Greengrass Key Concepts

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

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

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

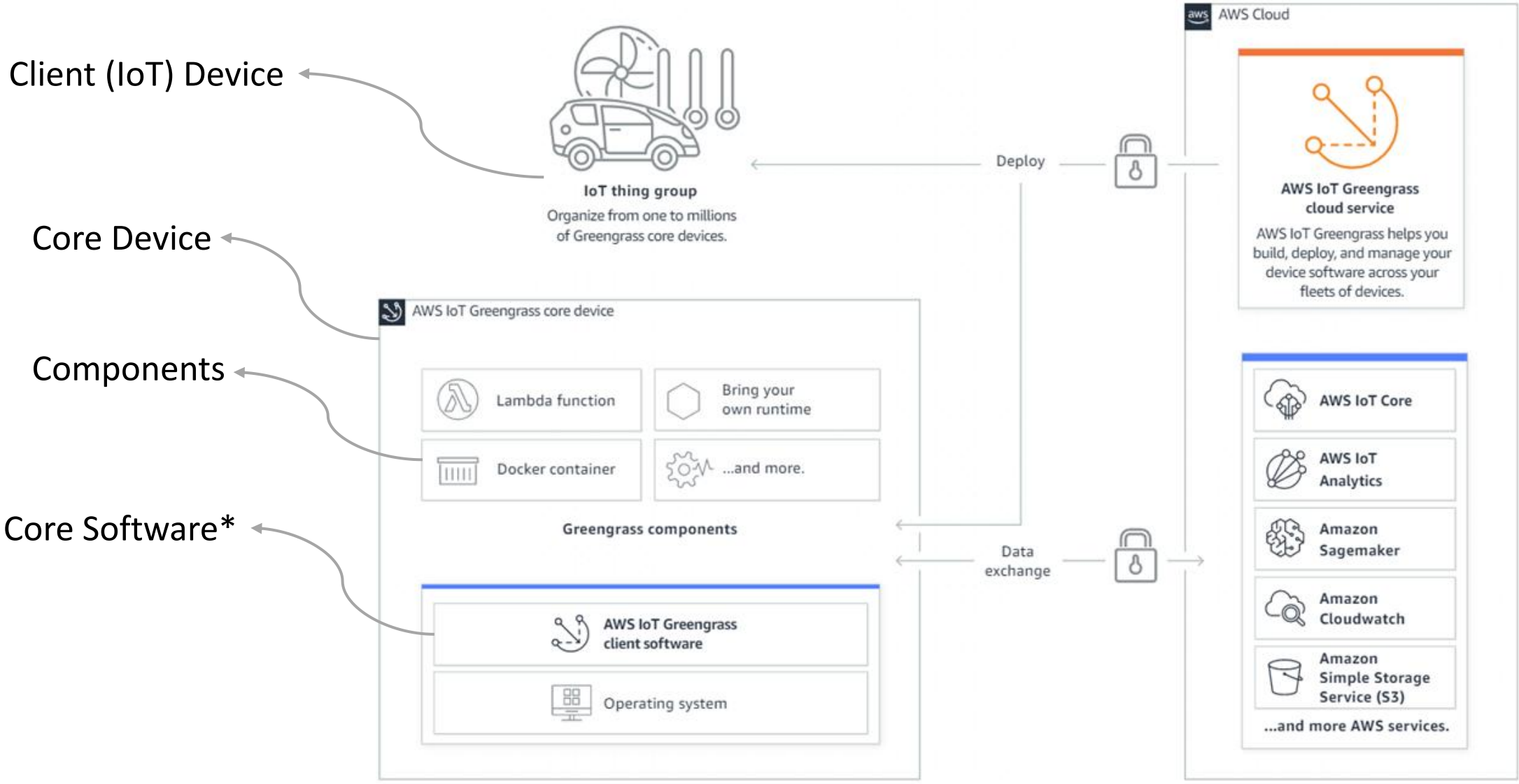
Deployment: The process of installing and configuring Greengrass components on a Greengrass Core device. This typically involves uploading the component code and configuration files to the device.



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

Physical Device Software Concept Action/Process

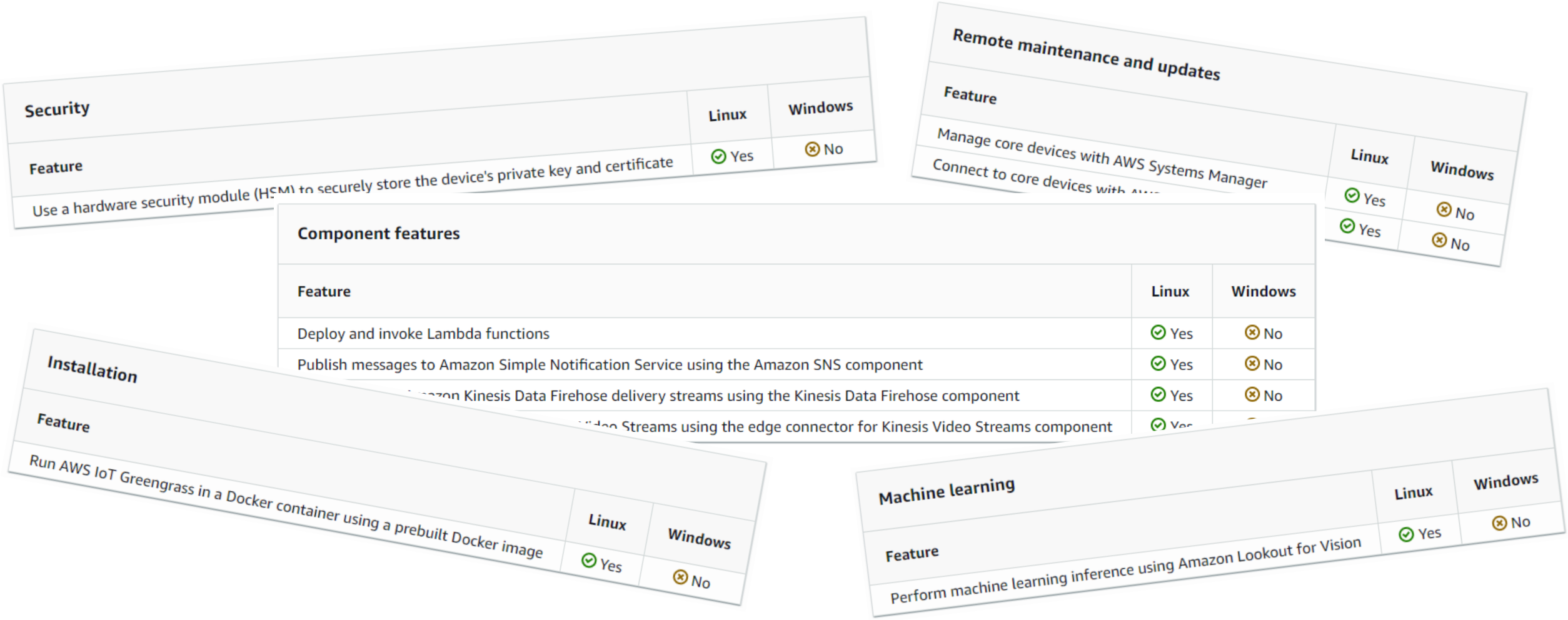
How AWS IoT Greengrass works



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

Selecting Client's Operating System

- Some features are supported on only certain operating systems!



Security

Feature	Linux	Windows
Use a hardware security module (HSM) to securely store the device's private key and certificate	Yes	No

Remote maintenance and updates

Feature	Linux	Windows
Manage core devices with AWS Systems Manager	Yes	No
Connect to core devices with AWS	Yes	No

Component features

Feature	Linux	Windows
Deploy and invoke Lambda functions	Yes	No
Publish messages to Amazon Simple Notification Service using the Amazon SNS component	Yes	No
Amazon Kinesis Data Firehose delivery streams using the Kinesis Data Firehose component	Yes	No
Video Streams using the edge connector for Kinesis Video Streams component	Yes	

Installation

Feature	Linux	Windows
Run AWS IoT Greengrass in a Docker container using a prebuilt Docker image	Yes	No

Machine learning

Feature	Linux	Windows
Perform machine learning inference using Amazon Lookout for Vision	Yes	No

Full feature list: https://docs.aws.amazon.com/greengrass/v2/developerguide/operating-system-feature-support-matrix.html

AWS IoT Greengrass Core Software

AWS IoT Greengrass Core Software

- The AWS IoT Greengrass Core software extends AWS functionality onto an AWS IoT Greengrass core device
- The AWS IoT Greengrass Core software provides a lot of functionality:
  - Deploy and invoke components and Lambda functions.
  - Support MQTT messaging between AWS IoT and components.
  - Interact with local devices that connect and communicate over MQTT.
  - Execution of ML models on edge devices, enabling low-latency, real-time analytics.
  - Efficient and reliable high-volume IoT data transfer to the AWS Cloud using Stream Manager.
  - Perform secure, over-the-air (OTA) software updates.
  - Provide secure, encrypted storage of local secrets.
  - And more…

Steps for Greengrass Core SW Installation

1. Set up your environment and install the required tools
2. Create temporary AWS credentials (using a temporary user)
3. Set up a Greengrass Core device in the AWS Management Console
4. Install the Greengrass Core sw using the temporary user's credentials
5. Remove the temporary user after installation

Step 1: Set Up Your Environment

For Linux Based Devices

- AWS IoT Greengrass Core software requires Java version 8 or higher.
- Amazon recommends to use OpenJDK 11 or Amazon Corretto 11.

For Debian-based or Ubuntu-based distributions:

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

For Red Hat-based distributions:

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

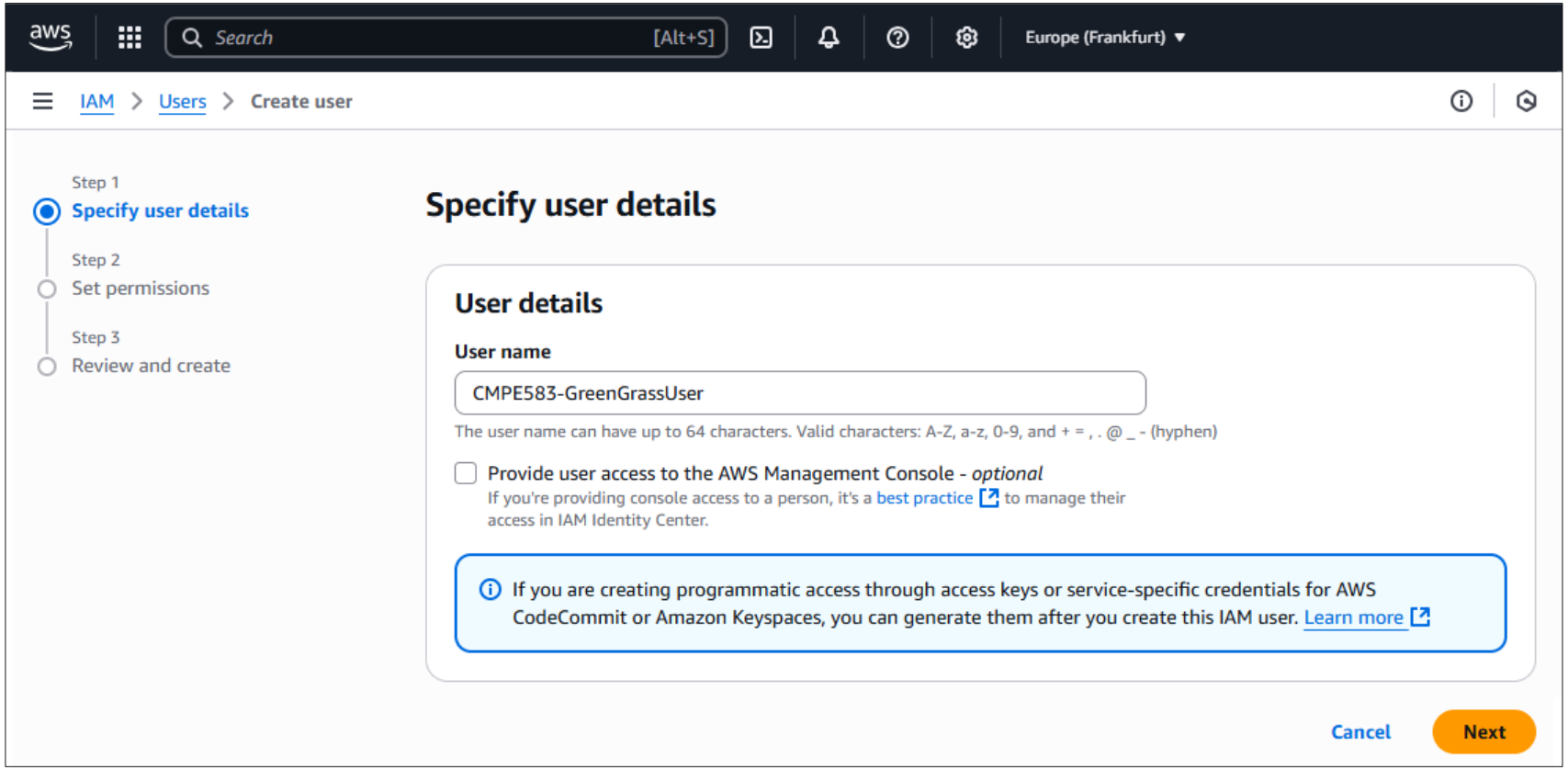
For Amazon Linux 2023 (AL2023):

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

Step 2: Create AWS Credentials

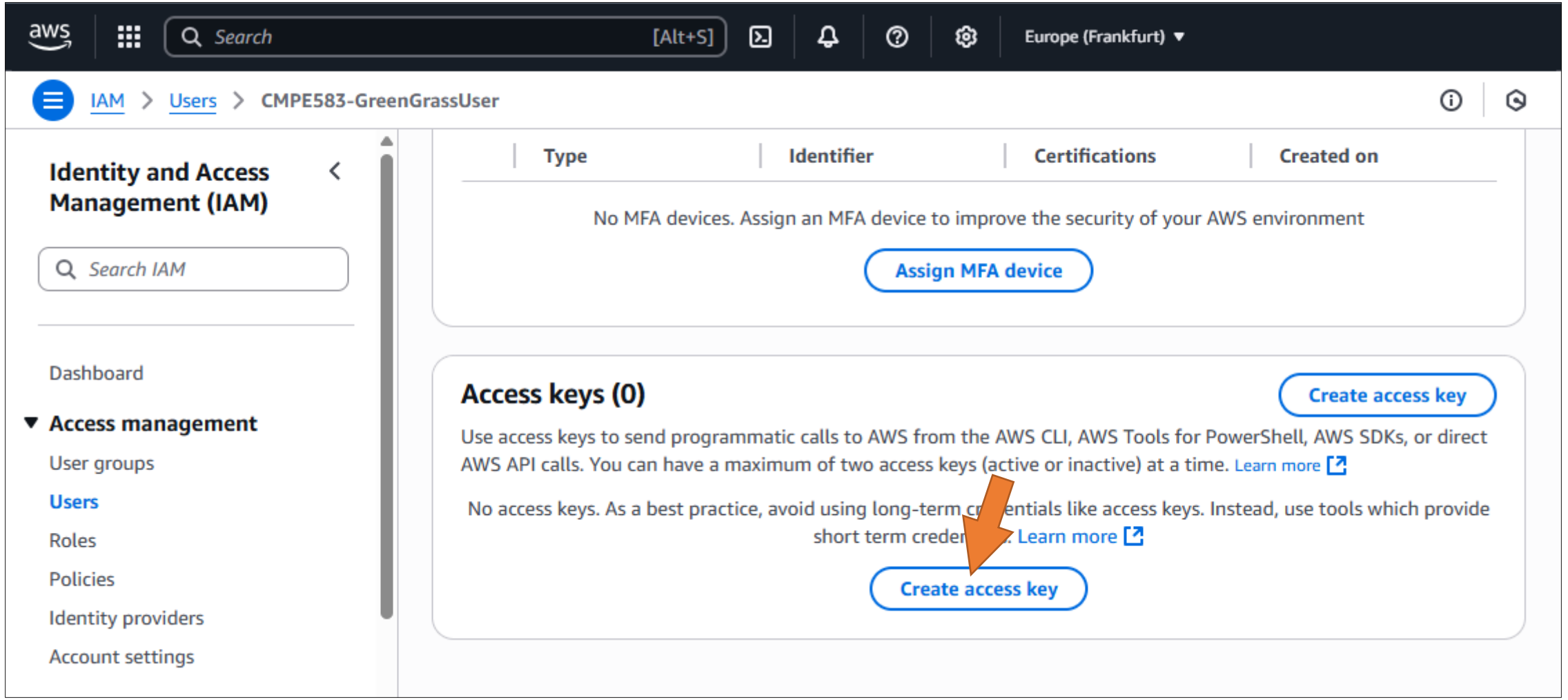
Create a Temporary User

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



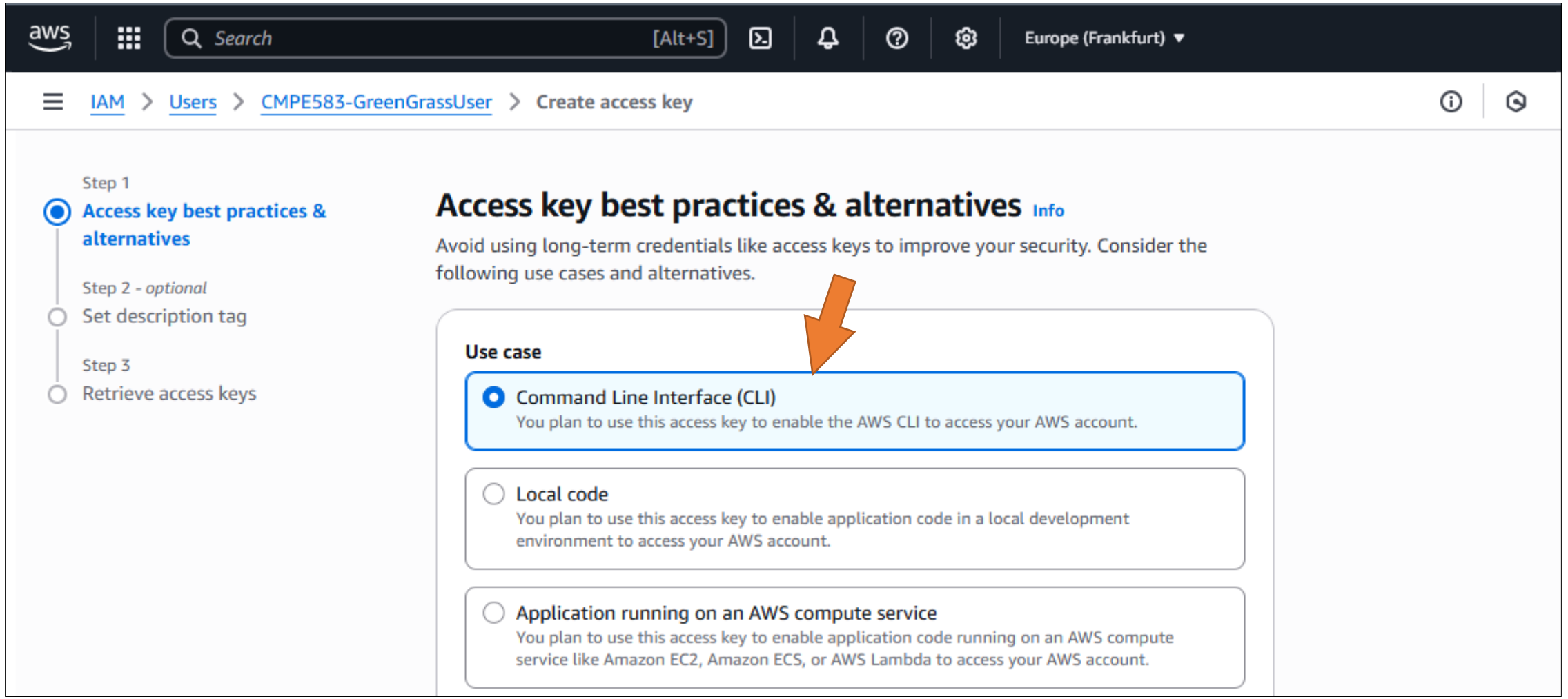
Step 2: Create AWS Credentials

Create Access Keys for the User



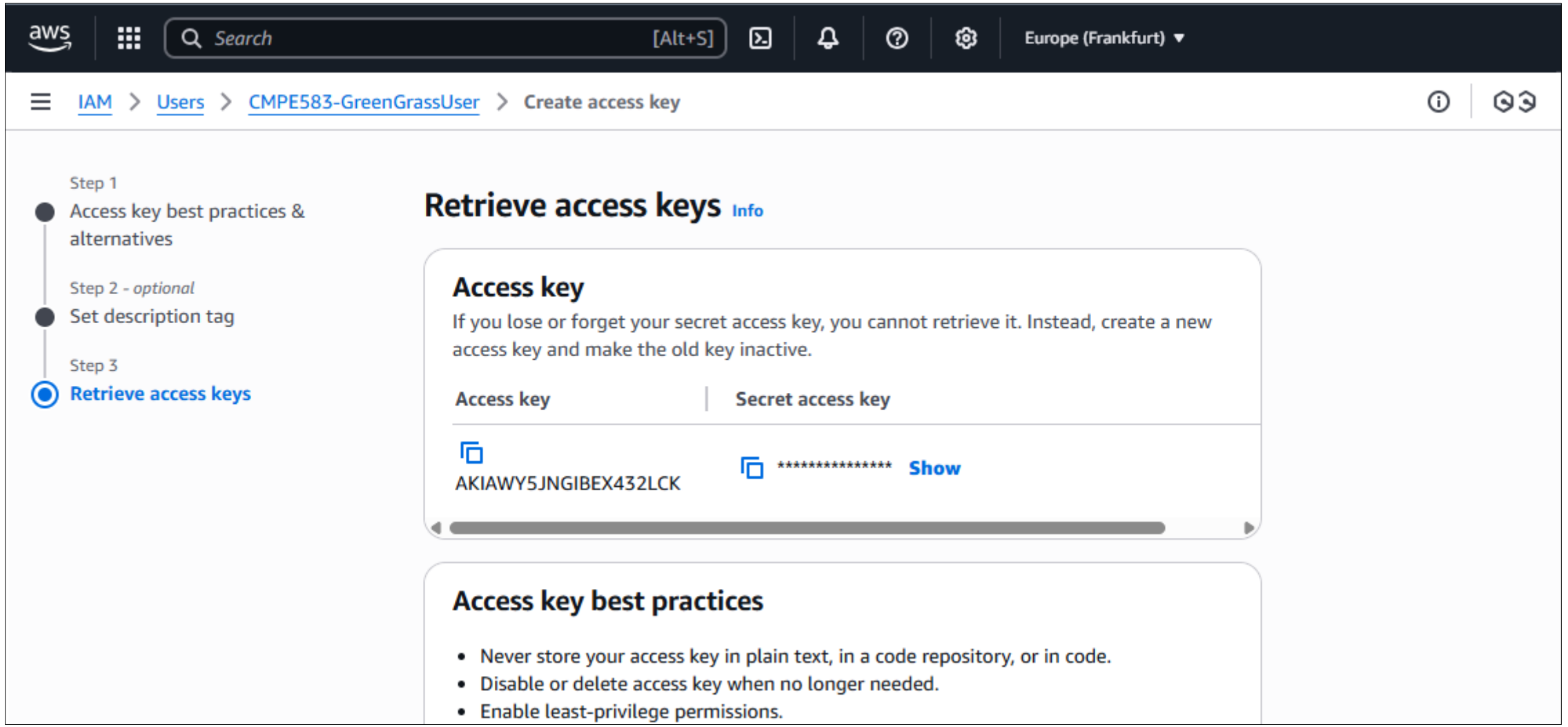
Step 2: Create AWS Credentials

Define Access Key Use-Case

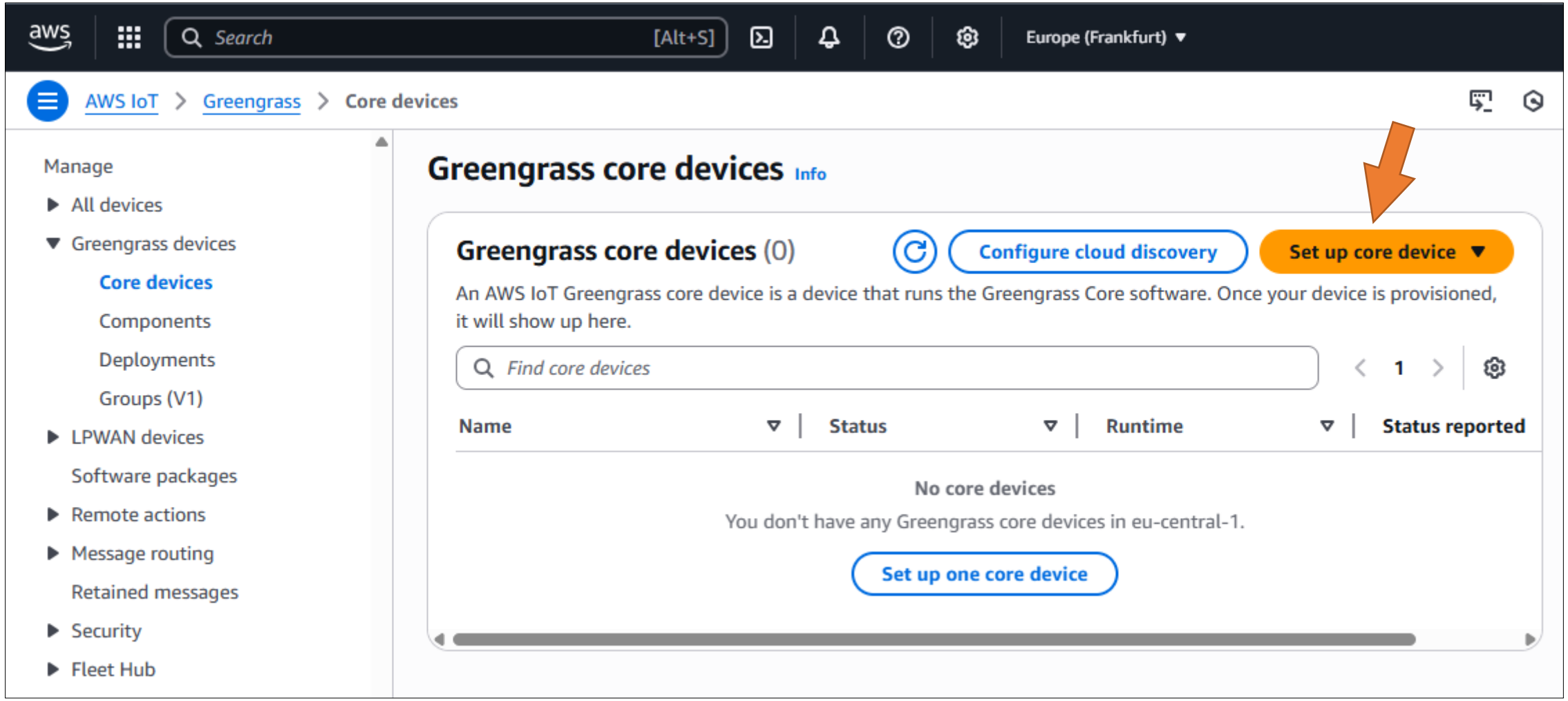


Step 2: Create AWS Credentials

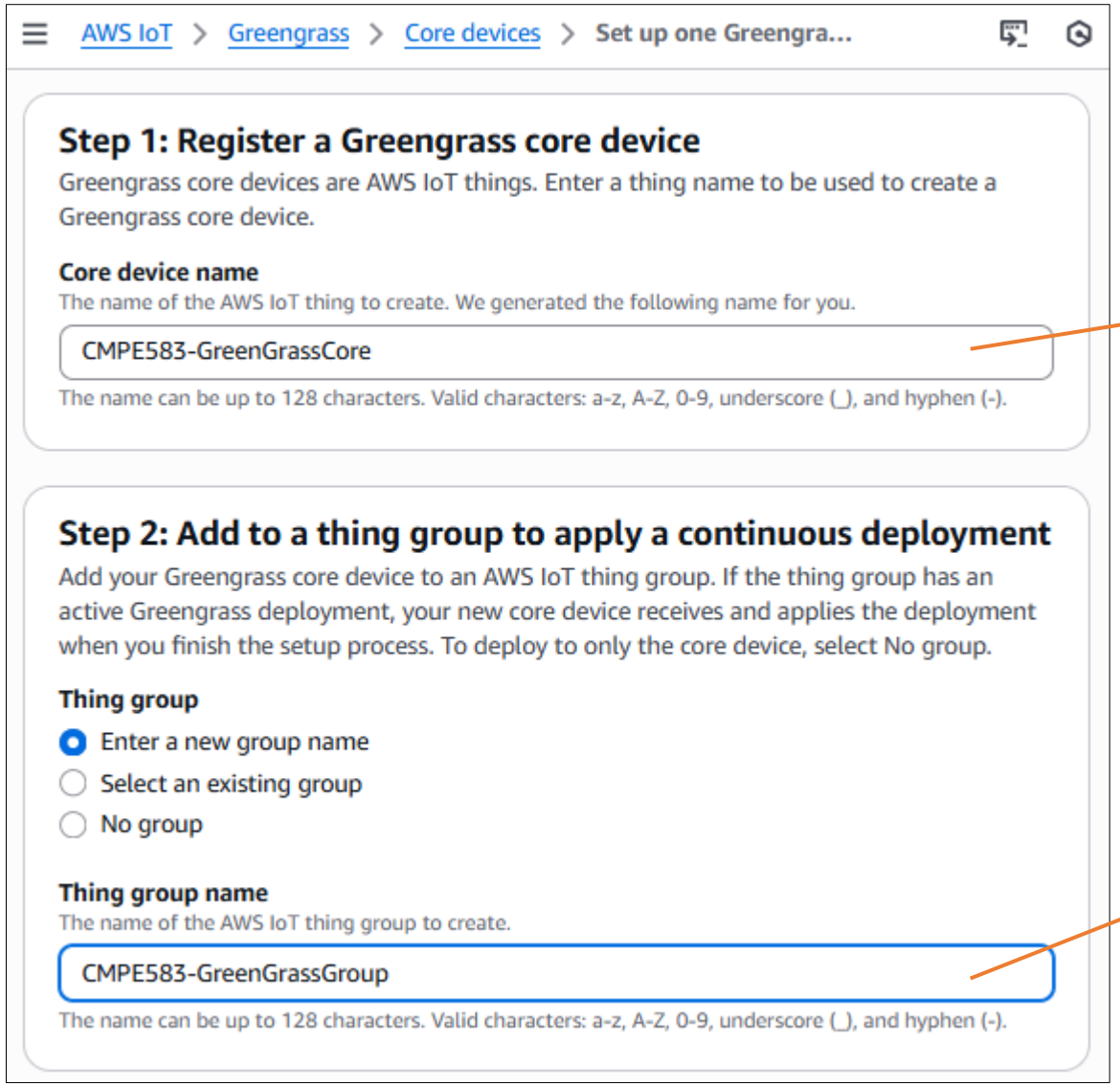
Copy the Access Keys



Step 3: Set Up a Greengrass Core Device in AWS



Step 3: Set Up a Greengrass Core Device in AWS

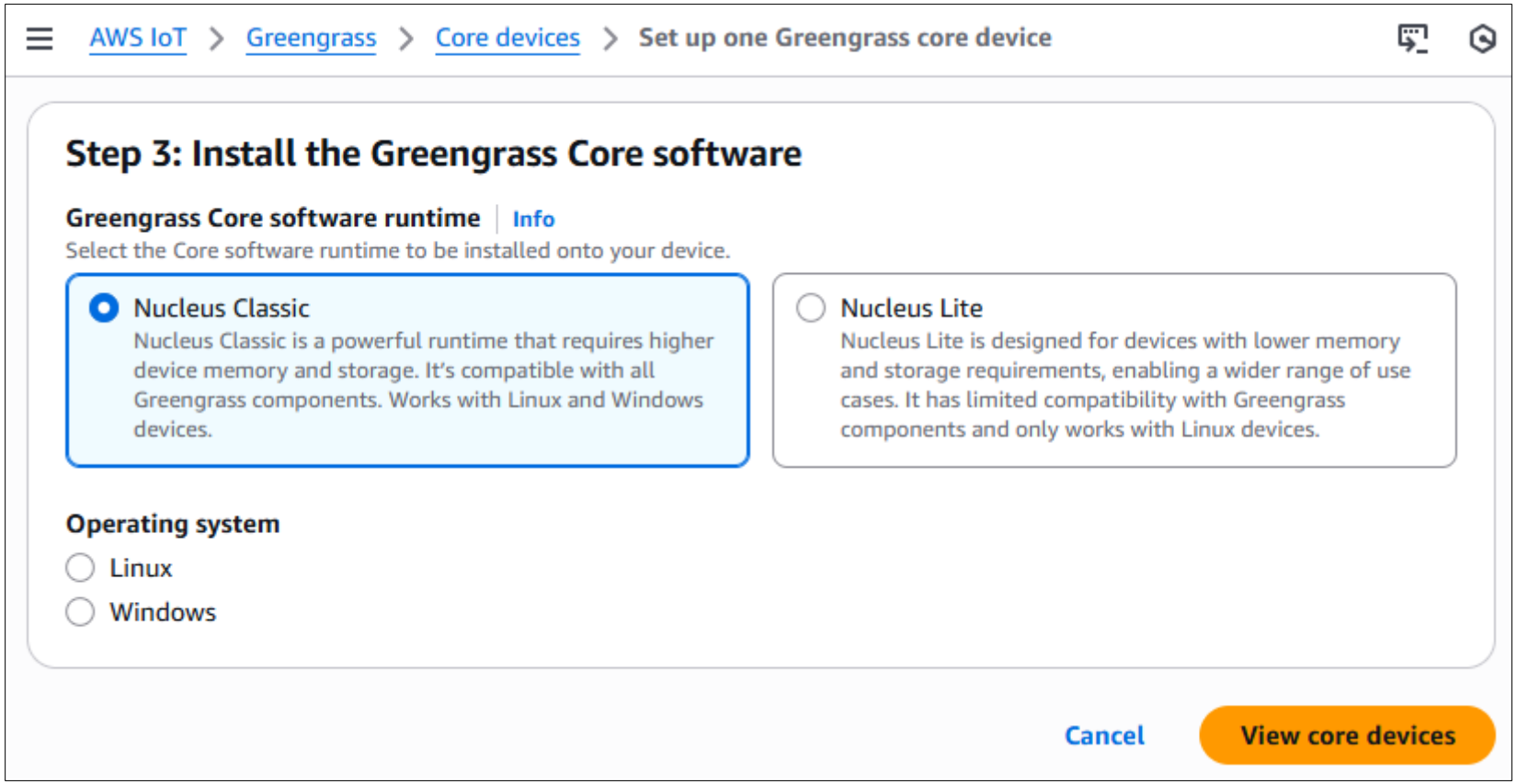


The name of the AWS IoT thing for your Greengrass core device.

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

Step 3: Set Up a Greengrass Core Device in AWS

Select Your Runtime



Step 4: Install the Greengrass Core Software

Configure AWS Credentials on the Client

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

  Provide credentials as environment variables before running the installer:

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

- Credentials are not saved by the Greengrass installer!

Step 4: Install the Greengrass Core Software

For Linux Based Devices

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

Download the installer:

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

Run the installer:

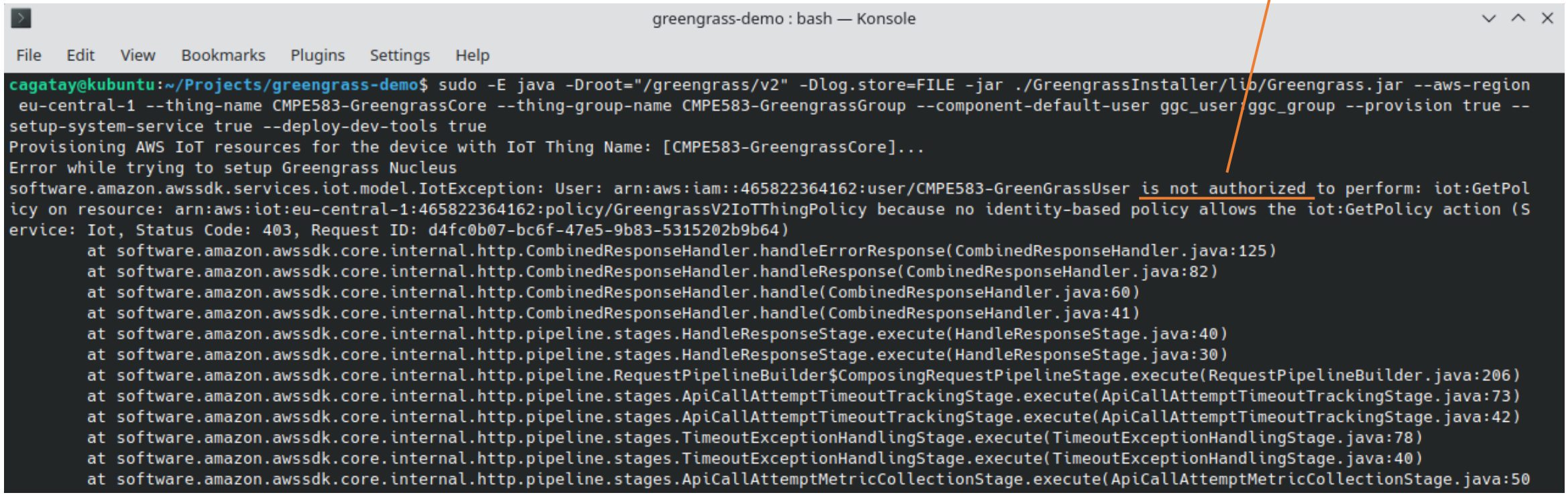
```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE -jar ./GreengrassInstaller/lib/Greengrass.jar --aws-
    region eu-central-1 --thing-name CMPE583-GreenGrassCore --thing-group-name CMPE583-
    GreenGrassGroup --component-default-user ggc_user:ggc_group --provision true --setup-system-service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

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

Step 4: Install the Greengrass Core Software

Run the Installer Command

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

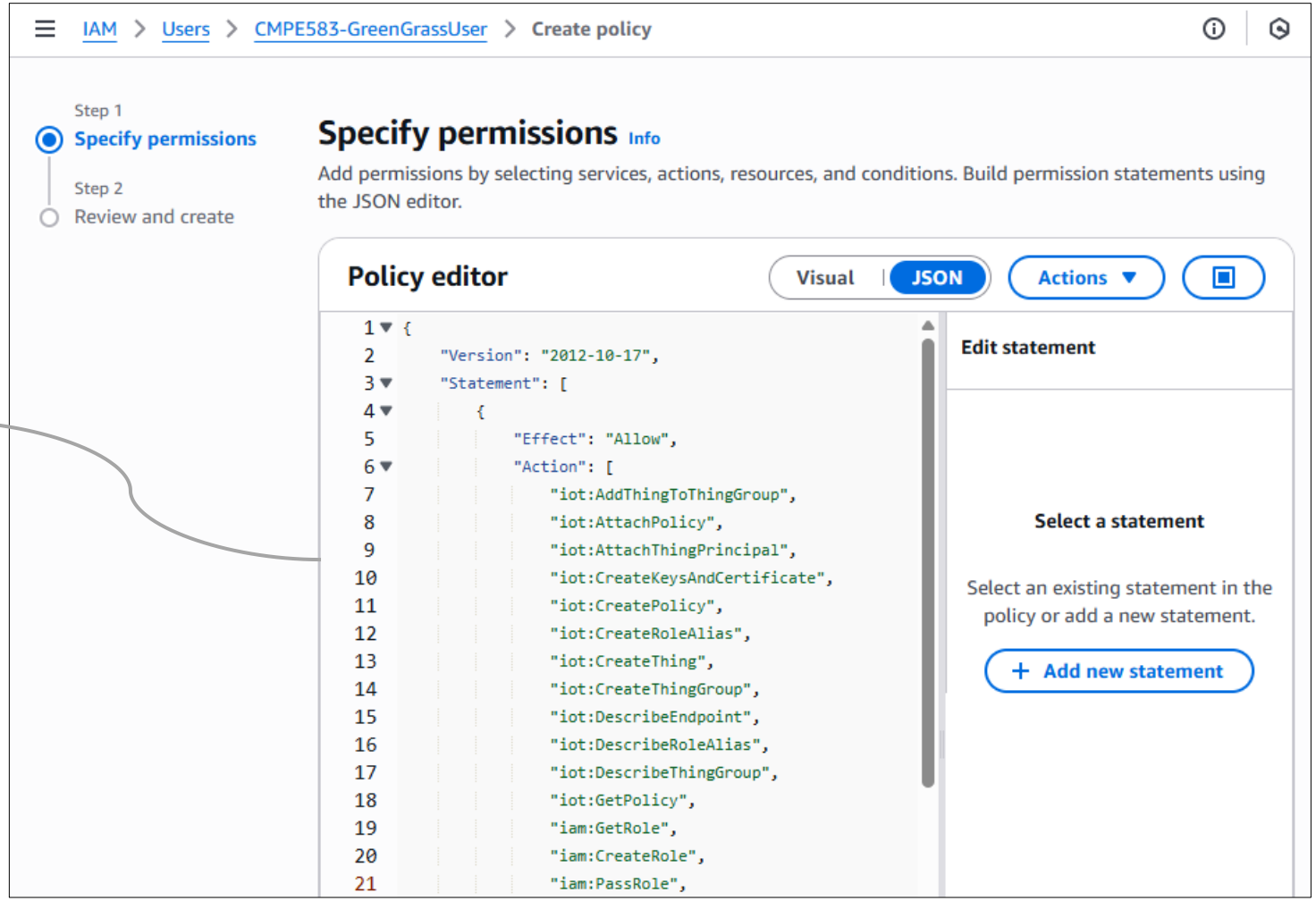


Step 4: Install the Greengrass Core Software

Set Required Permissions to Greengrass User

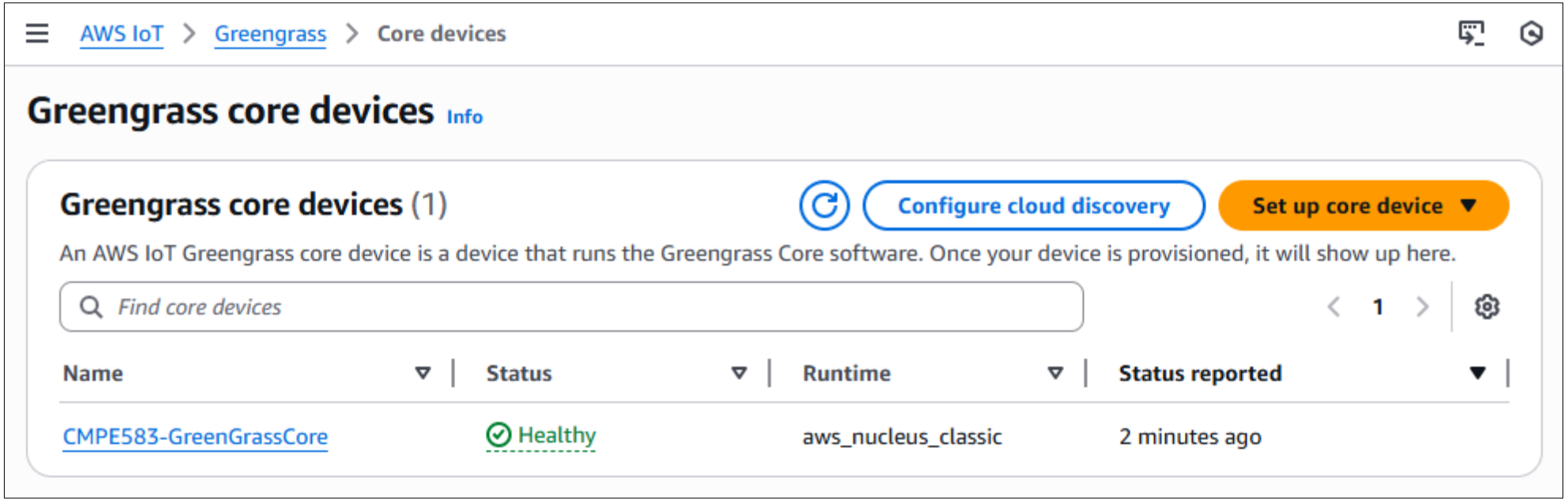
- When installing a core device, the user we created needs certain permissions to provision AWS IoT resources.

Permission content available in GitHub
- *AWS/IAM/cmpe583-GreengrassUser-ProvisionPolicy.json*



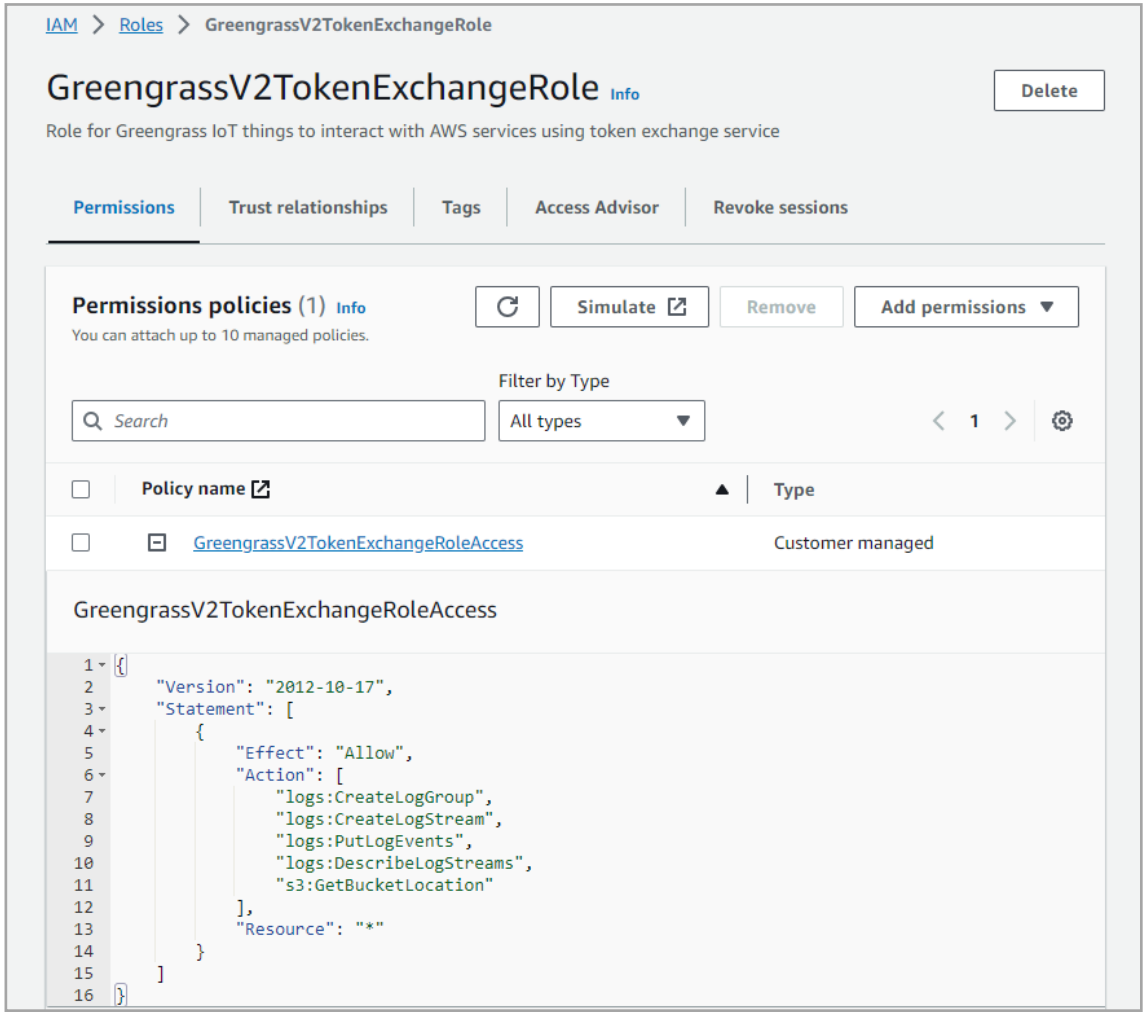
Step 4: Install the Greengrass Core Software

Rerun the Installer and Check If Core Device Is Healthy



Step 4: Install the Greengrass Core Software

Check If GreengrassV2TokenExchangeRole Is Created



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

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

Step 4: Install the Greengrass Core software

Troubleshooting

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

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

Step 4: Install the Greengrass Core software

Monitor AWS IoT Greengrass logs

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

```
/greengrass/v2
└── logs
    ├── greengrass.log
    ├── greengrass_2021_09_14_15_0.log
    ├── ComponentName.log
    ├── ComponentName_2021_09_14_15_0.log
    └── main.log
```

AWS IoT Greengrass root folder

The AWS IoT Greengrass Core software rotates log files every hour or when they exceed a file size limit

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

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

Step 4: Install the Greengrass Core Software

Auto Start of Greengrass Core Software

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

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

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

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

Uninstall the AWS IoT Greengrass Core Software

For Linux Based Devices

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

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

- Verify that the service is deleted.

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

- Remove the root folder from the device.

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

Greengrass Components

Greengrass Components

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

Recipe File

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

Artifacts

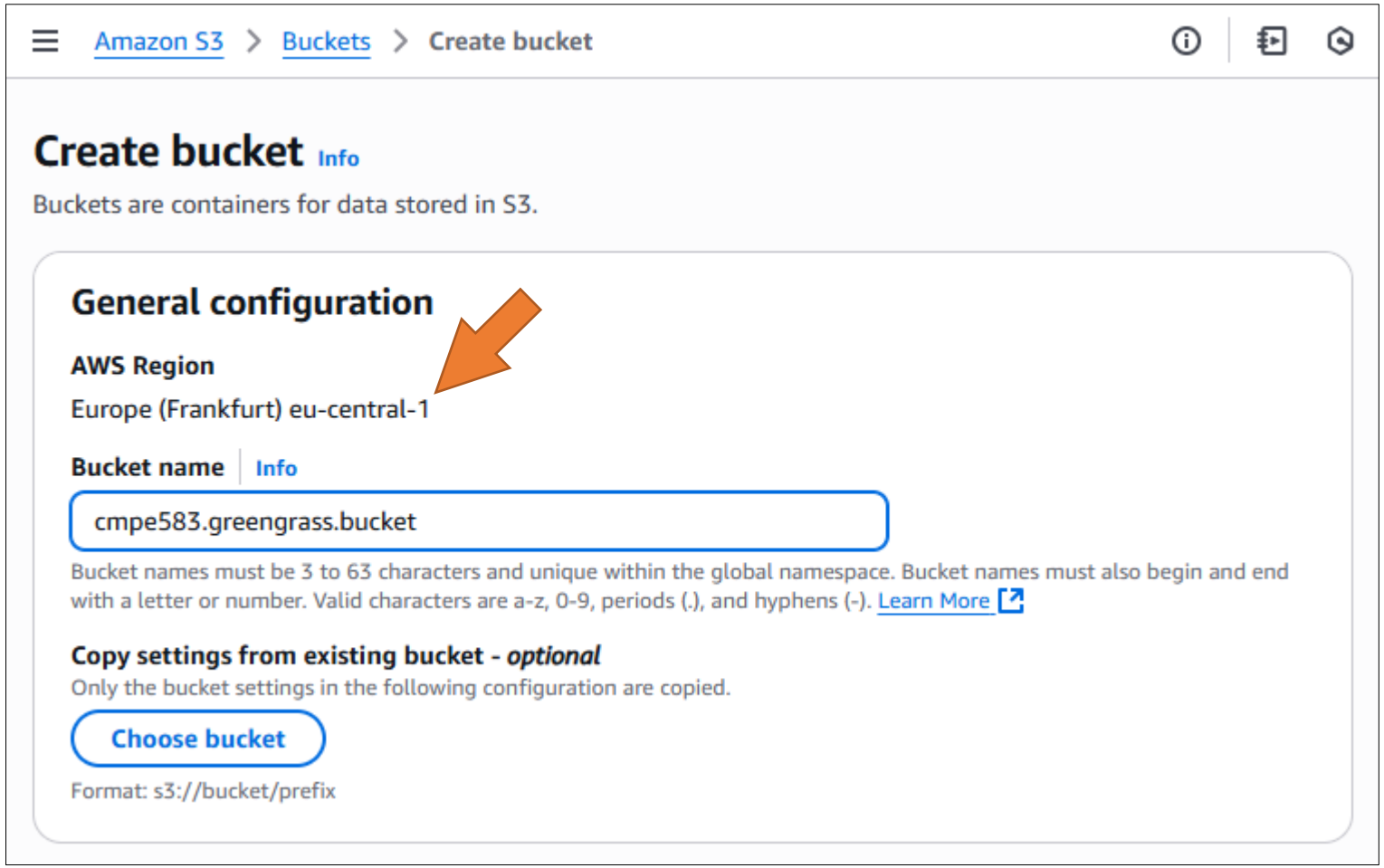
- Artifacts are component binaries.
- Components can have any number of artifacts.
- Artifacts can include:
  - Scripts
  - Compiled code
  - Static resources
  - Other files that a component consumes

Steps for Greengrass Component Deployement

1. Upload artifacts to Amazon S3
2. Set S3 access policies
3. Create the component and configure its recipe
4. Deploy the component via the AWS Management Console
5. Verify deployment in the console and on the device terminal

Step 1: Upload the Artifacts

Create a bucket via AWS S3 (Console)

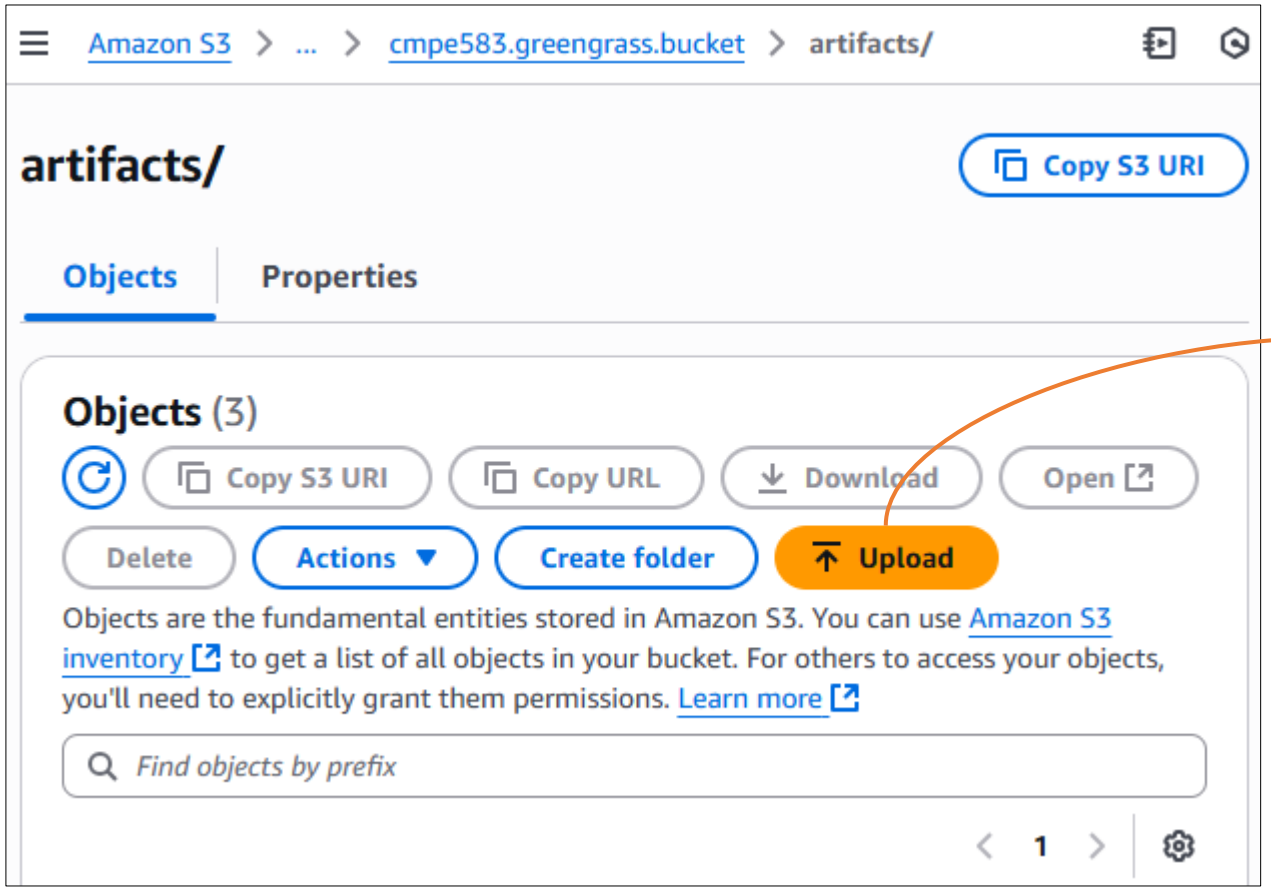


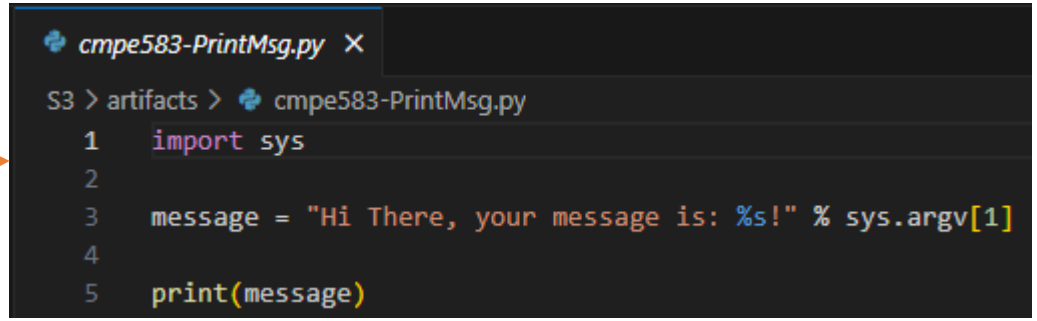
- Be sure that the S3 bucket is in the same AWS Region where you create the component.
- AWS IoT Greengrass doesn't support cross-Region requests for component artifacts!

Step 1: Upload the Artifacts

Create artifacts folder & upload files

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





Step 2: Set Policies to Access Files in S3

Via AWS IAM (Console)

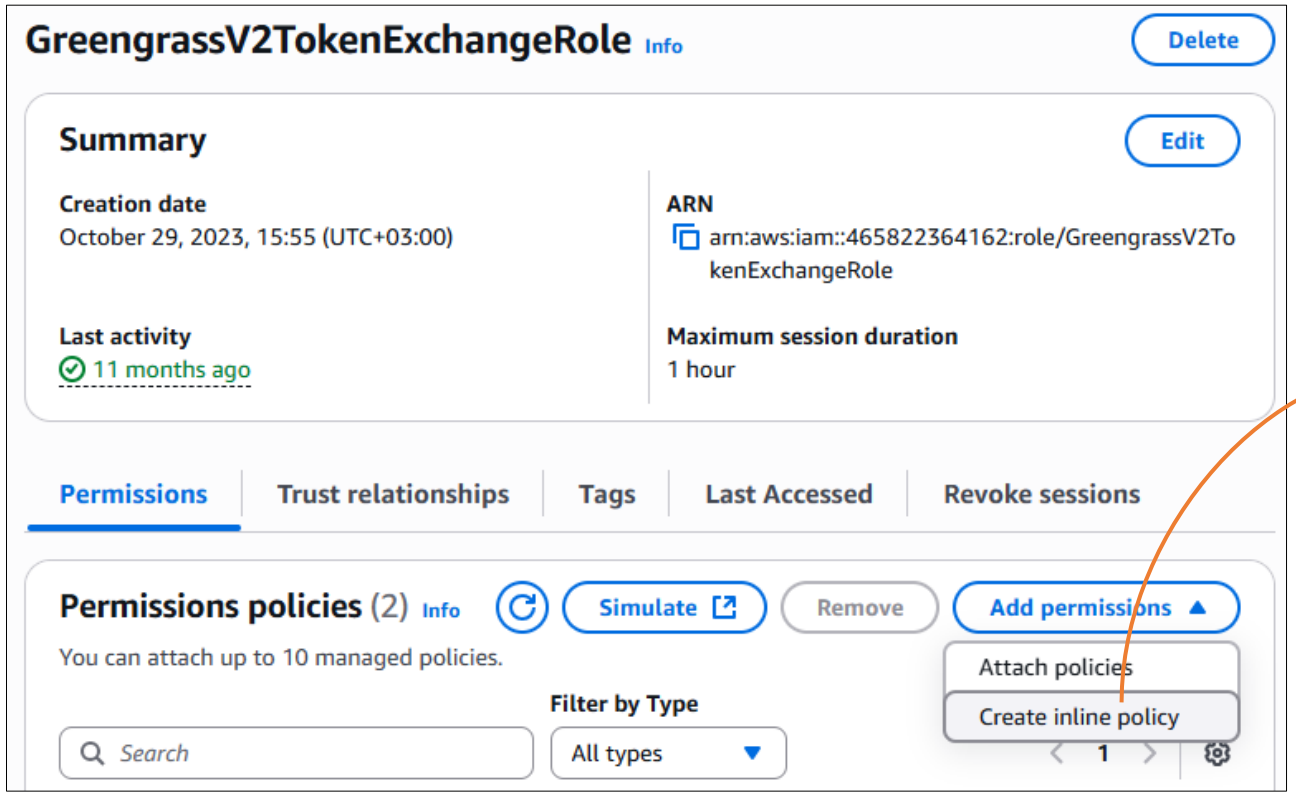
- Greengrass component artifacts should be stored in AWS S3 bucket.
- Therefore, you should allow the core device to access component artifacts in the S3 bucket.
  - Attach a policy similar to policy below to GreengrassV2TokenExchangeRole from AWS IAM console:

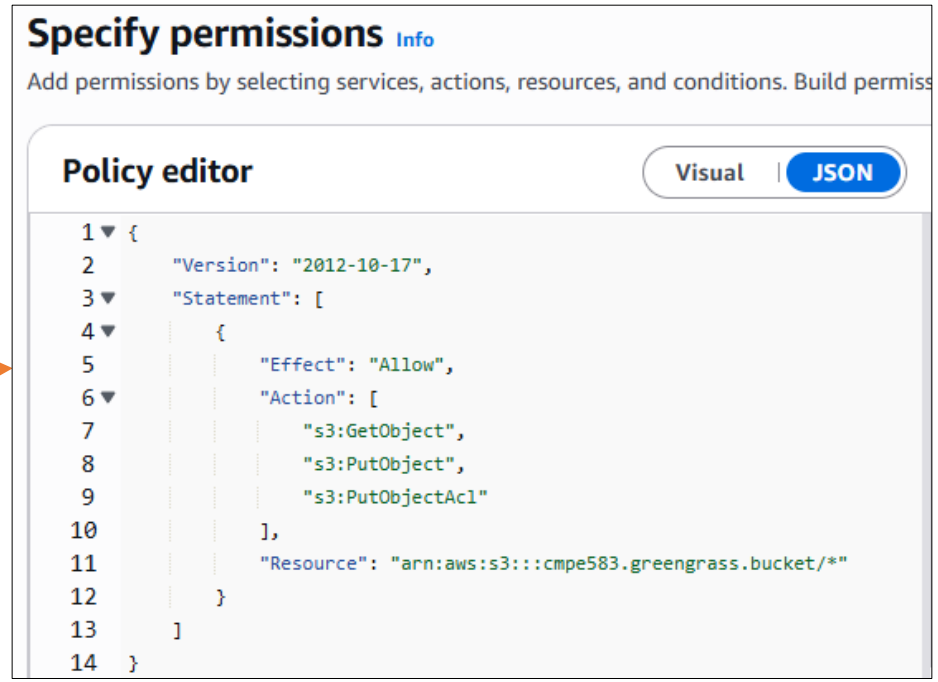
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::cmpe583.greengrass.bucket/*"
        }
    ]
}
```

Step 2: Set Policies to Access Files in S3

Via AWS IAM (Console)

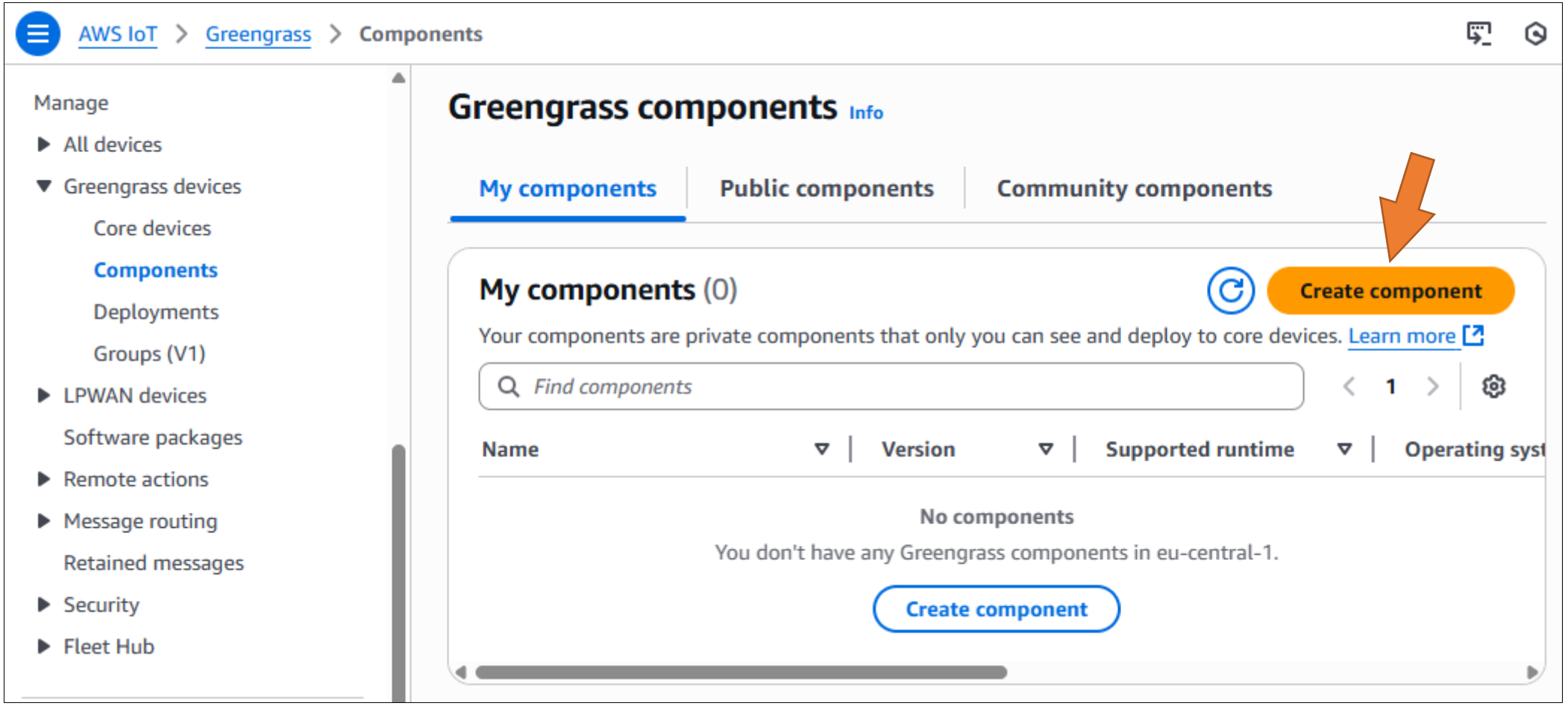
- You can create a new policy or simply attach an inline policy to related role in order to allow core device to Access objects in the S3 bucket.





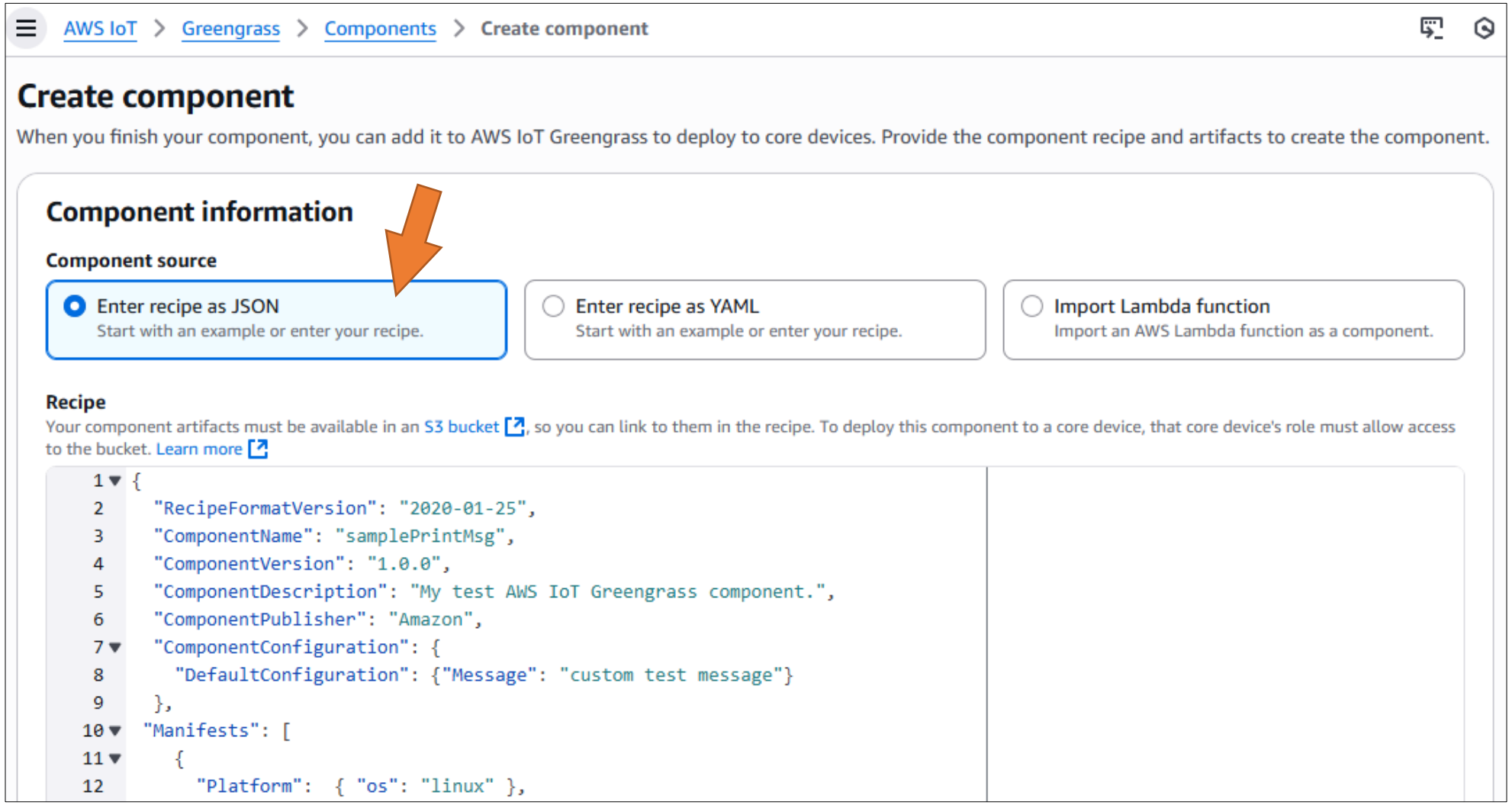
Step 3: Create the Component

Via AWS IoT Greengrass (Console)



Step 3: Create the Component

Define the Recipe



Step 3: Create the Component

Configure the Recipe

```
{
 "RecipeFormatVersion": "2020-01-25",
 "ComponentName": "samplePrintMsg",
 "ComponentVersion": "1.0.0",
 "ComponentDescription": "My test AWS IoT Greengrass component.",
 "ComponentPublisher": "Amazon",
 "ComponentConfiguration": {
  "DefaultConfiguration": {"Message": "custom test message"}
 },
 "Manifests": [
  {
   "Platform":  { "os": "linux" },
   "Lifecycle": { "run": "python3 -u {artifacts:path}/print_msg.py \"{configuration:/Message}\"" },
   "Artifacts": [ { "URI": "s3://cmpe583.greengrass.bucket/artifacts/print_msg.py" } ]
  }
 ]
}
```

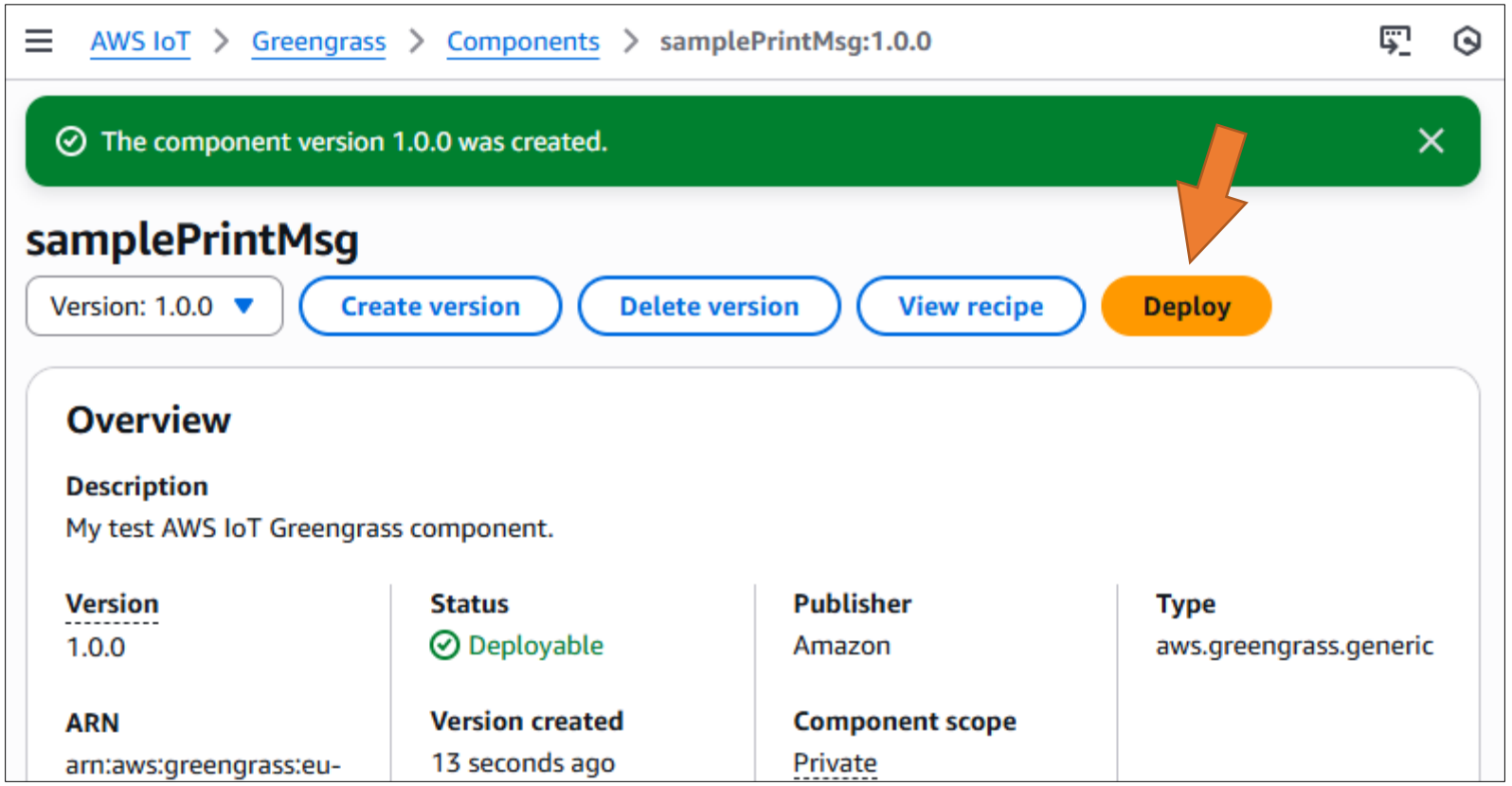
This test will be run on Linux device. So other OSs are ignored!

Define how to execute your artifact

Provide the artifact URI

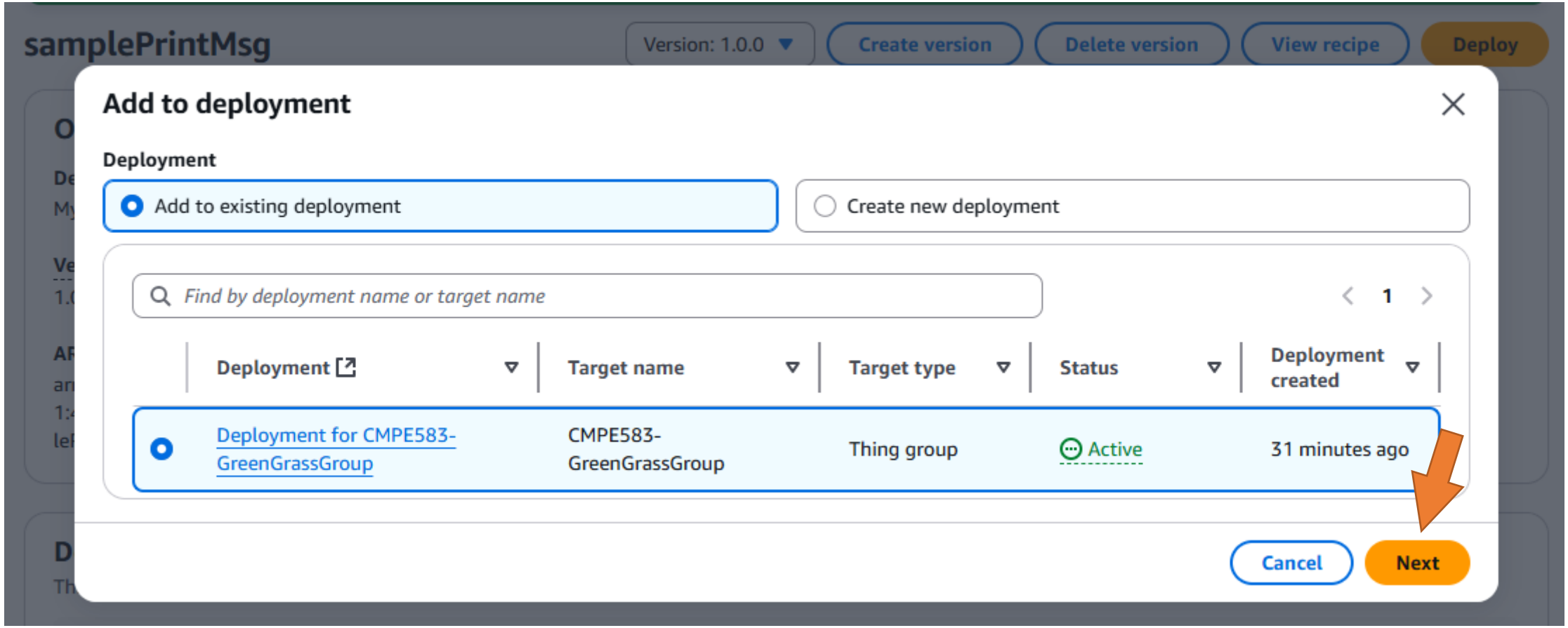
Step 4: Deploy the Component

Via AWS IoT Greengrass (console)



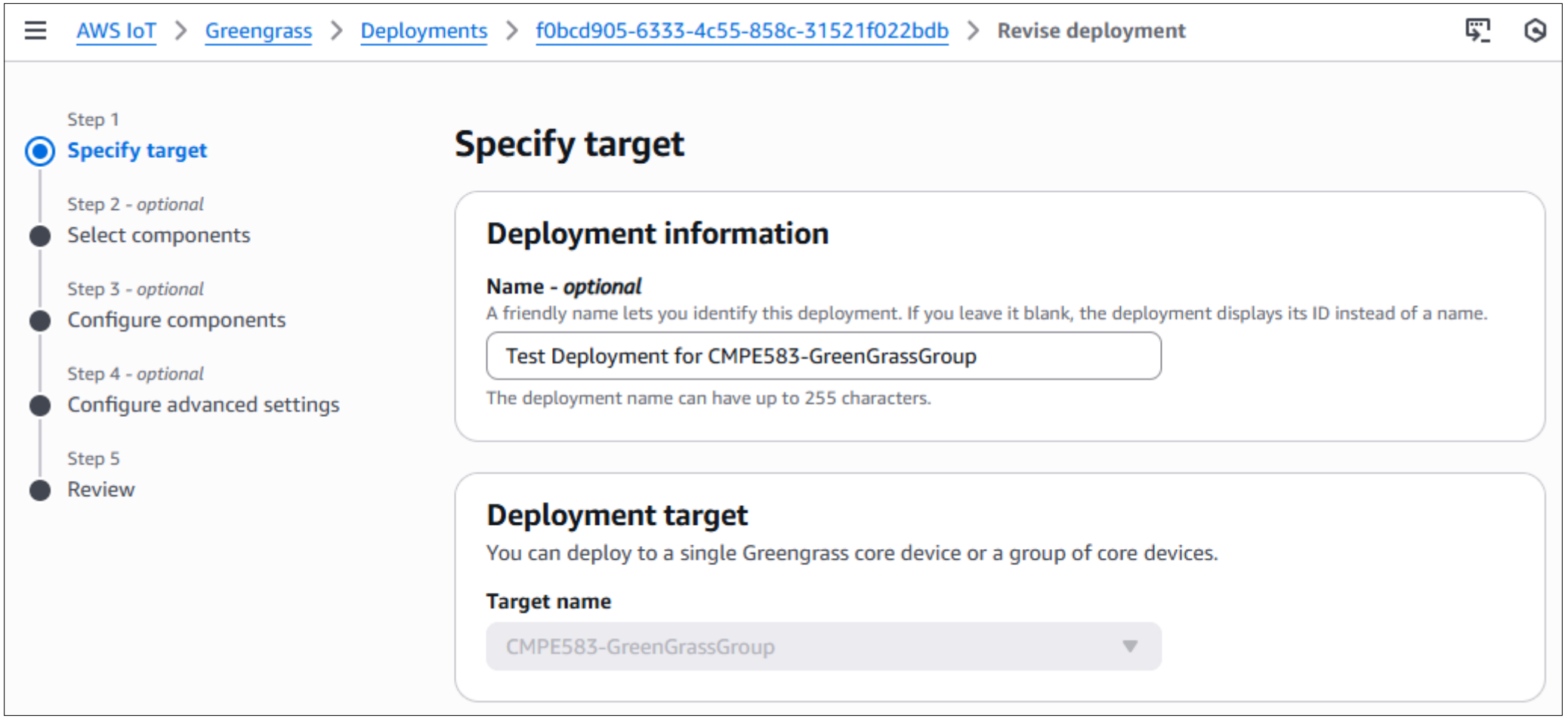
Step 4: Deploy the Component

Via AWS IoT Greengrass (console)



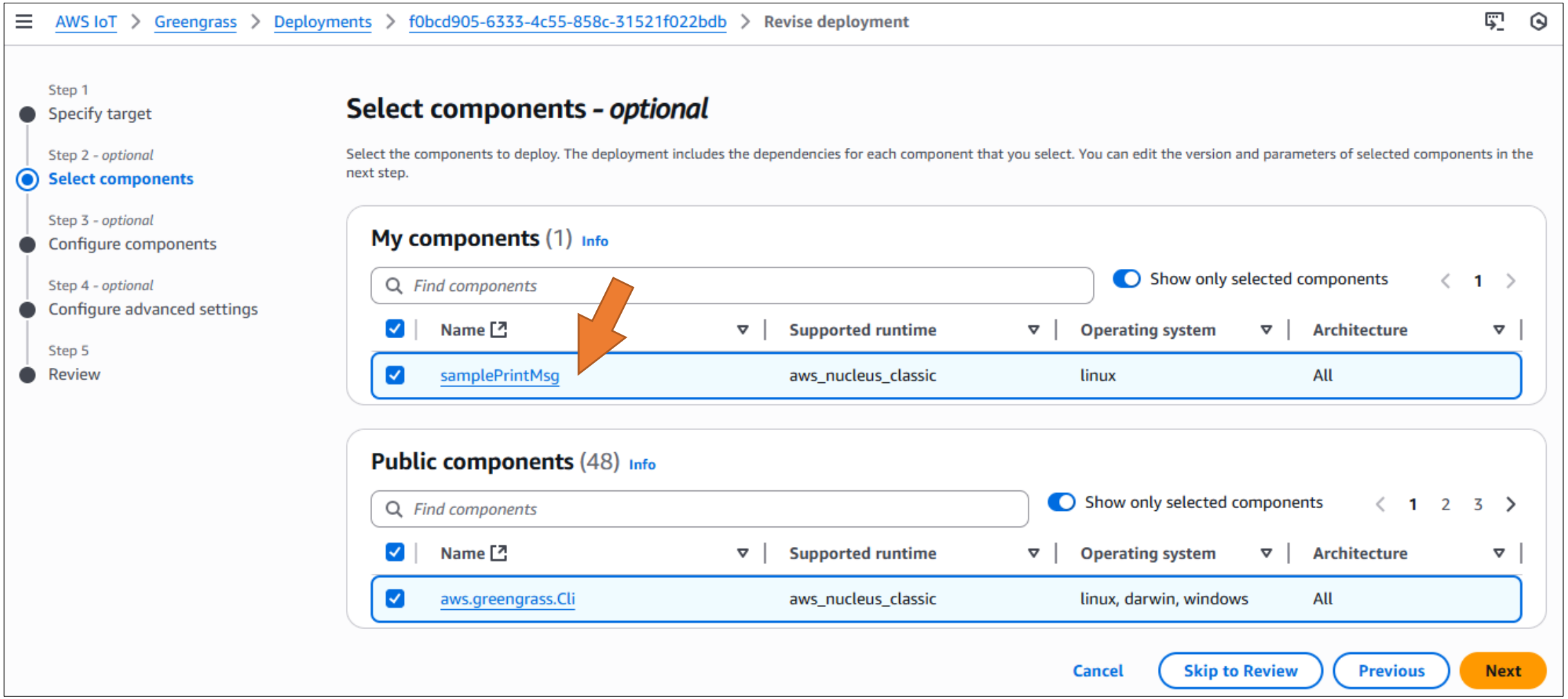
Step 4: Deploy the Component

Configure Deployment



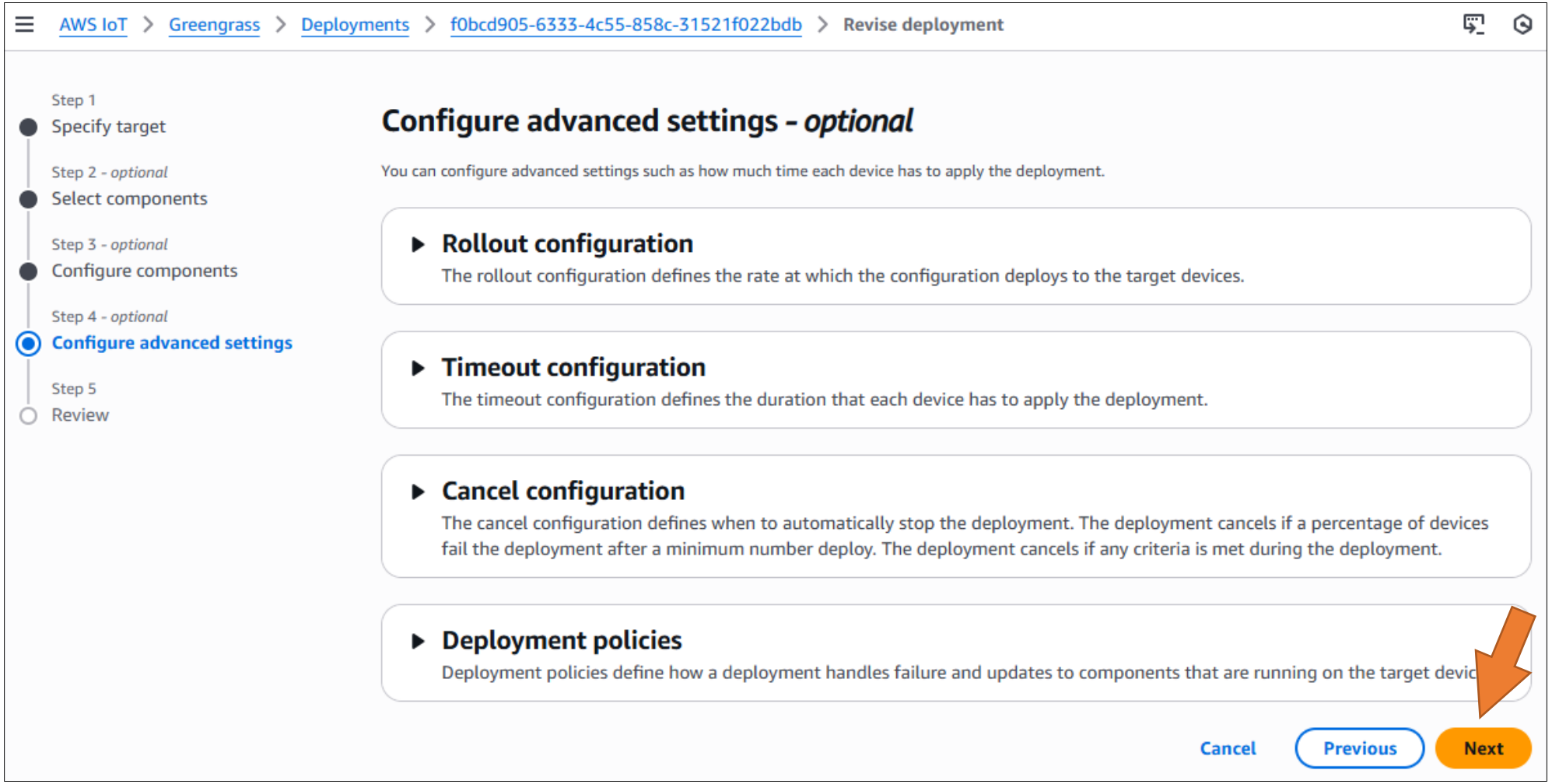
Step 4: Deploy the Component

Select the Component



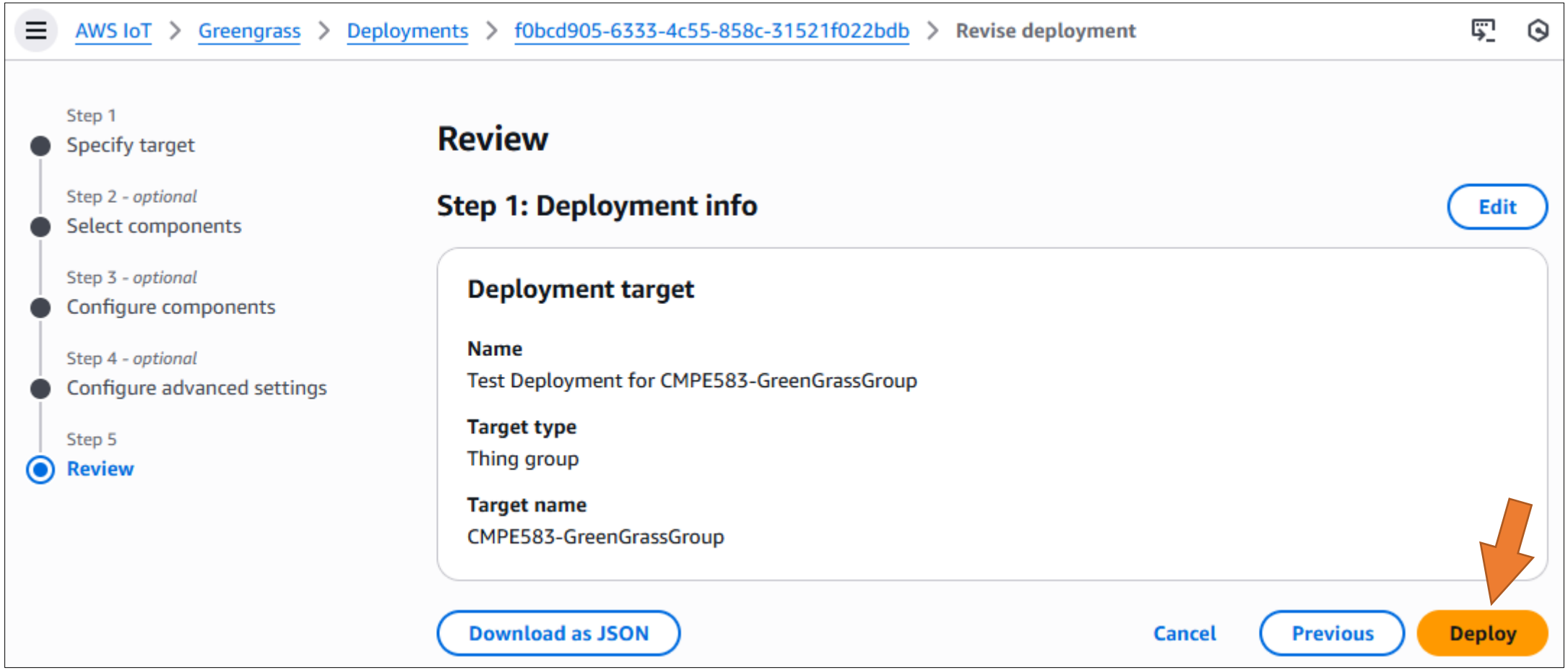
Step 4: Deploy the Component

Configure Advanced Settings



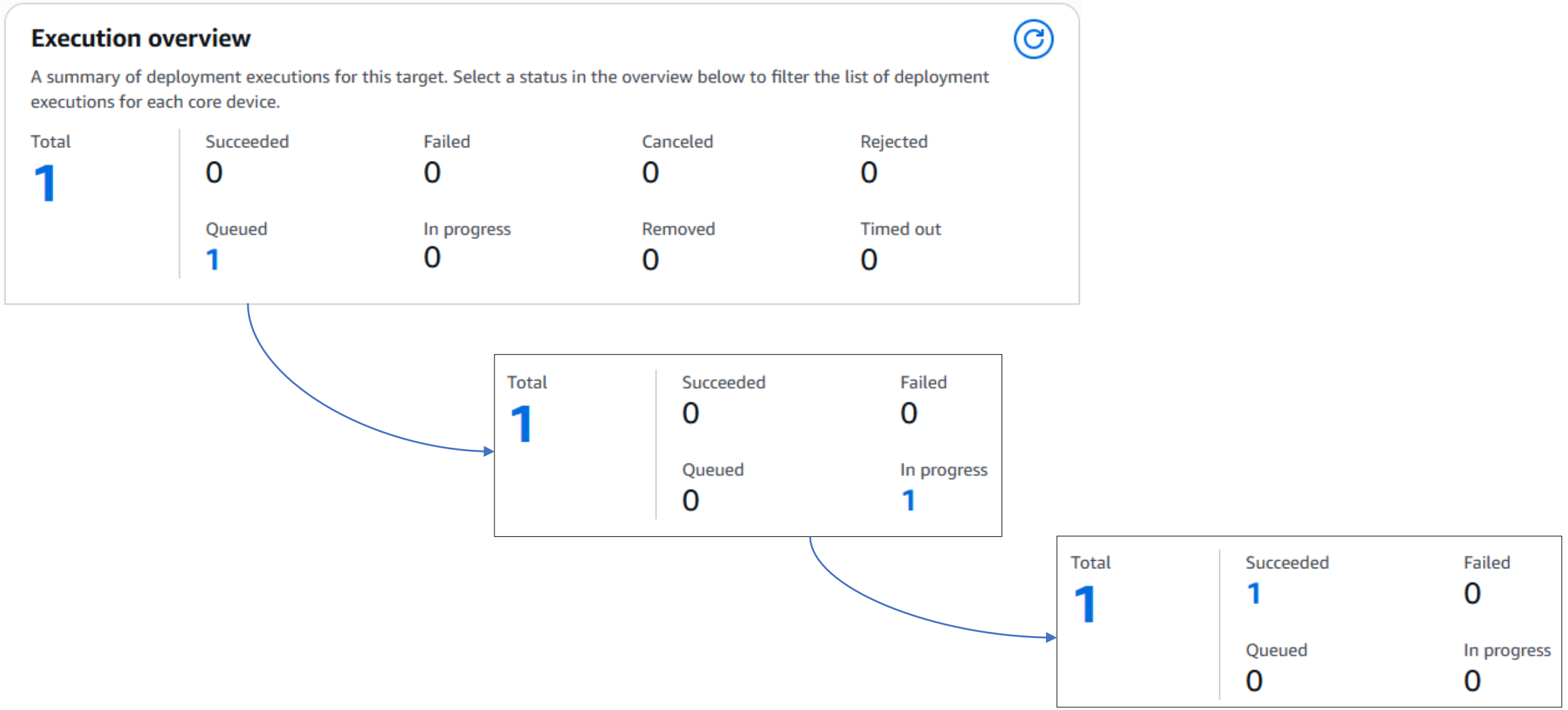
Step 4: Deploy the Component

Review & Deploy



Step 5: Verify the Component

Check AWS IoT Greengrass Console



Step 5: Verify the Component

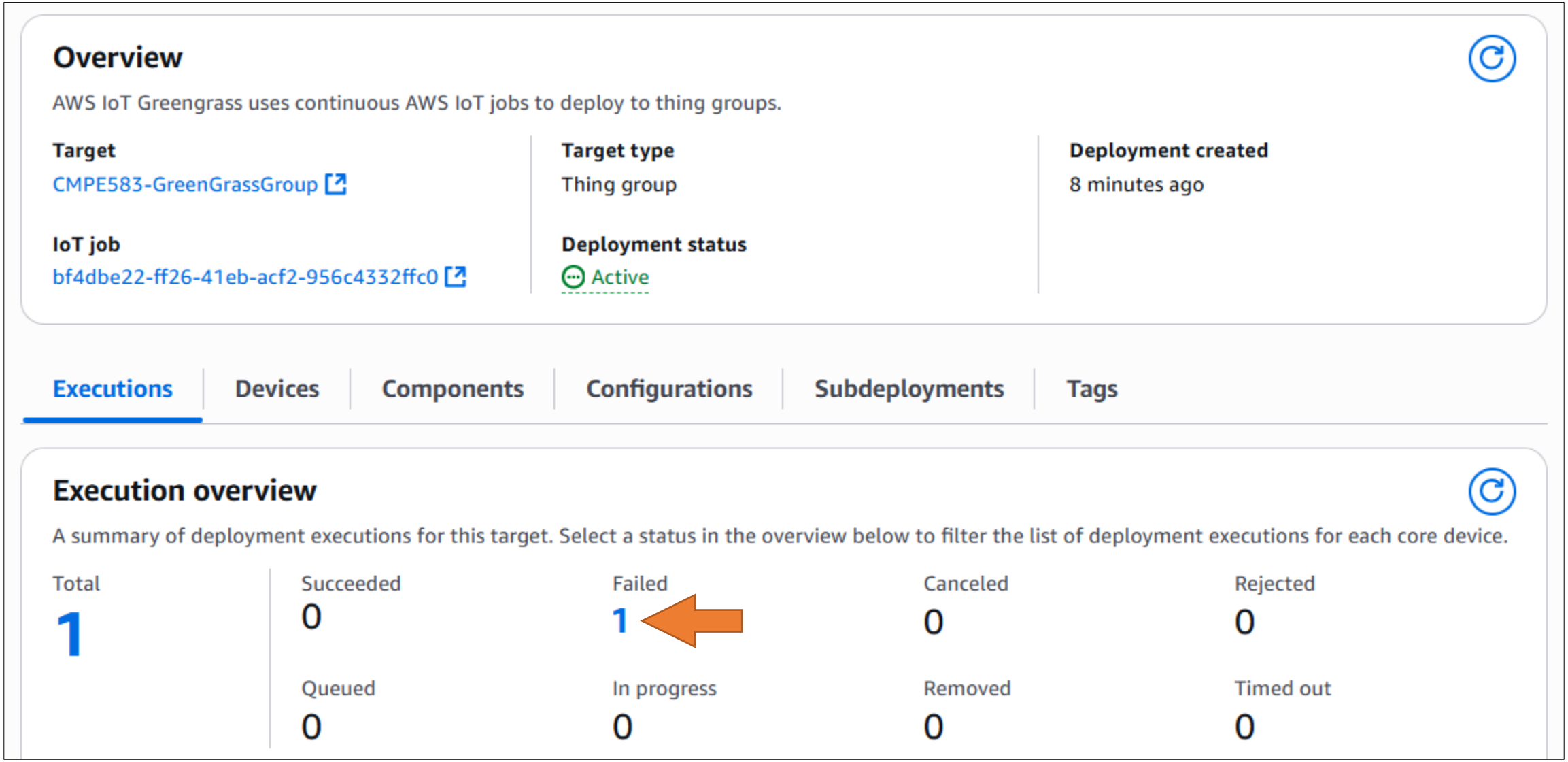
Check Log Files

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

```
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo# ls /greengrass/v2/logs/
aws.greengrass.Nucleus.log  greengrass_2023_10_29_15_0.log  greengrass.log  main.log  samplePrintMsg.log
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo#
root@kubuntu:/home/cagatay/Projects/greengrass-demo# 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.lif
ecycle.run, serviceName=samplePrintMsg, currentState=STARTING, command=["python3 -u /greengrass/v2/packages/artifacts/samplePr
intMsg/1.0.0/cmpe583-Prin..."]}
2023-10-29T13:04:00.305Z [INFO] (Copier) samplePrintMsg: stdout. Hi There, your message is: custom test message!. {scriptName=
services.samplePrintMsg.lifecycle.run, serviceName=samplePrintMsg, currentState=RUNNING}
2023-10-29T13:04:00.310Z [INFO] (Copier) samplePrintMsg: Run script exited. {exitCode=0, serviceName=samplePrintMsg, currentSt
ate=RUNNING}
```

All Good!

Troubleshooting



Troubleshooting

Check greengrass.log File

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

```
2023-09-23T10:40:57.822Z [INFO] (pool-2-thread-27) com.aws.greengrass.tes.CredentialRequestHandler: Received IAM credentials that will be cached until 2023-09-23
T11:35:57Z. {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.buckett/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: jlg8Vcxx
9IKjUwa/zzbgfRR/ujoTbrA9ZUoYZvrwqSQY7z8LN4vjtzdD+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.

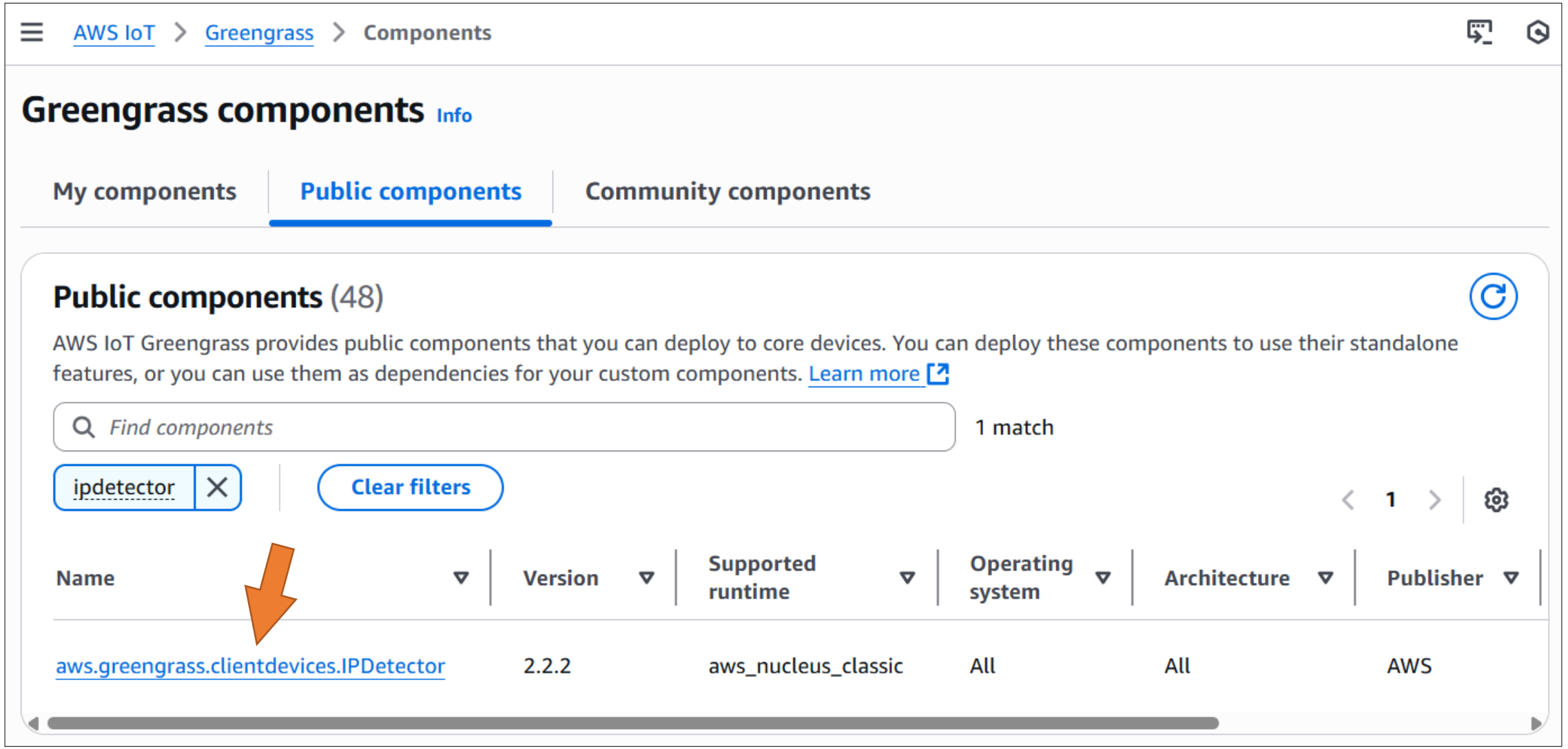
AWS Provided Components

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

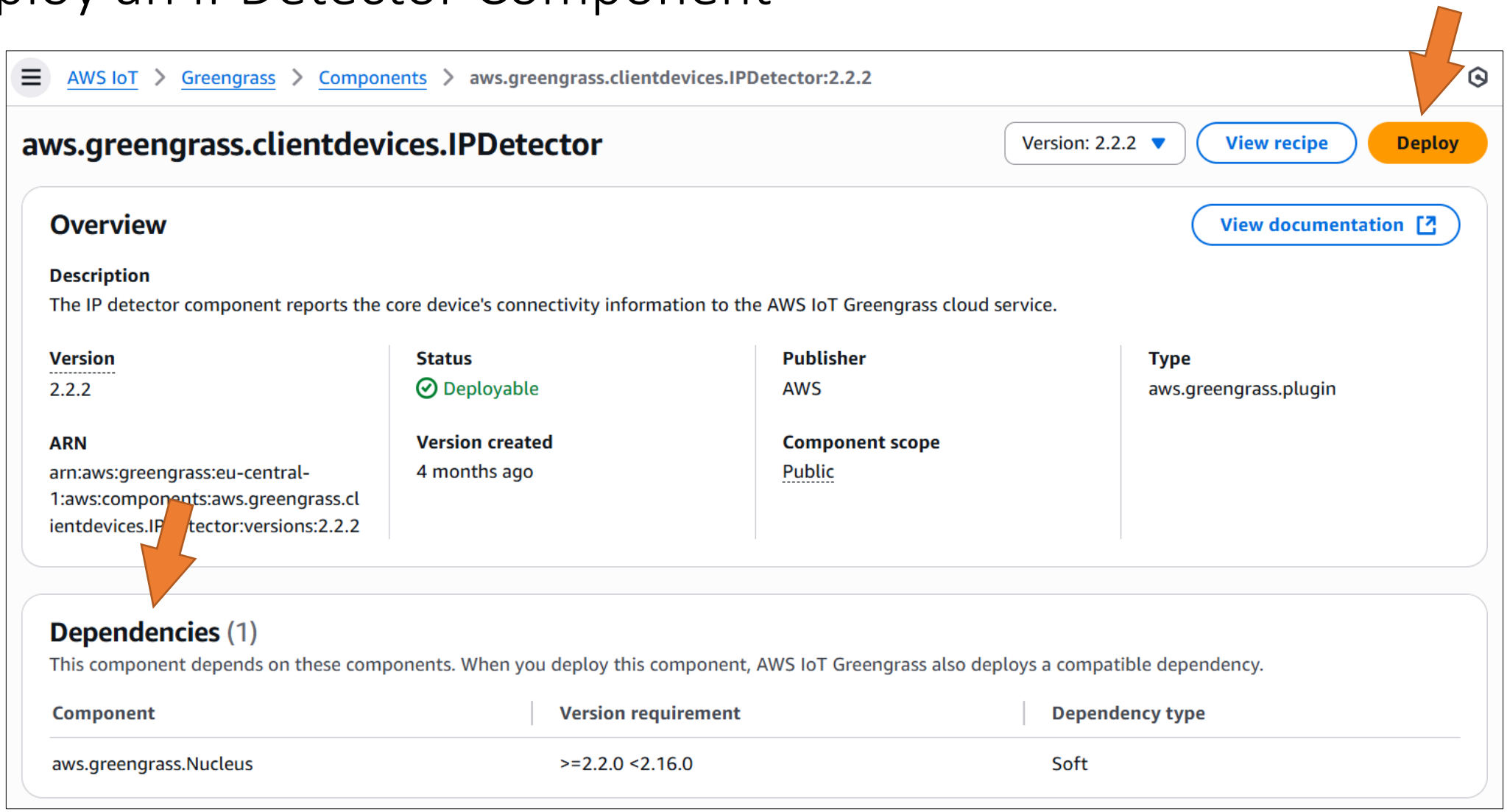
Collecting IP Addresses of Core Devices

IPDetector



Collecting IP Addresses of Core Devices

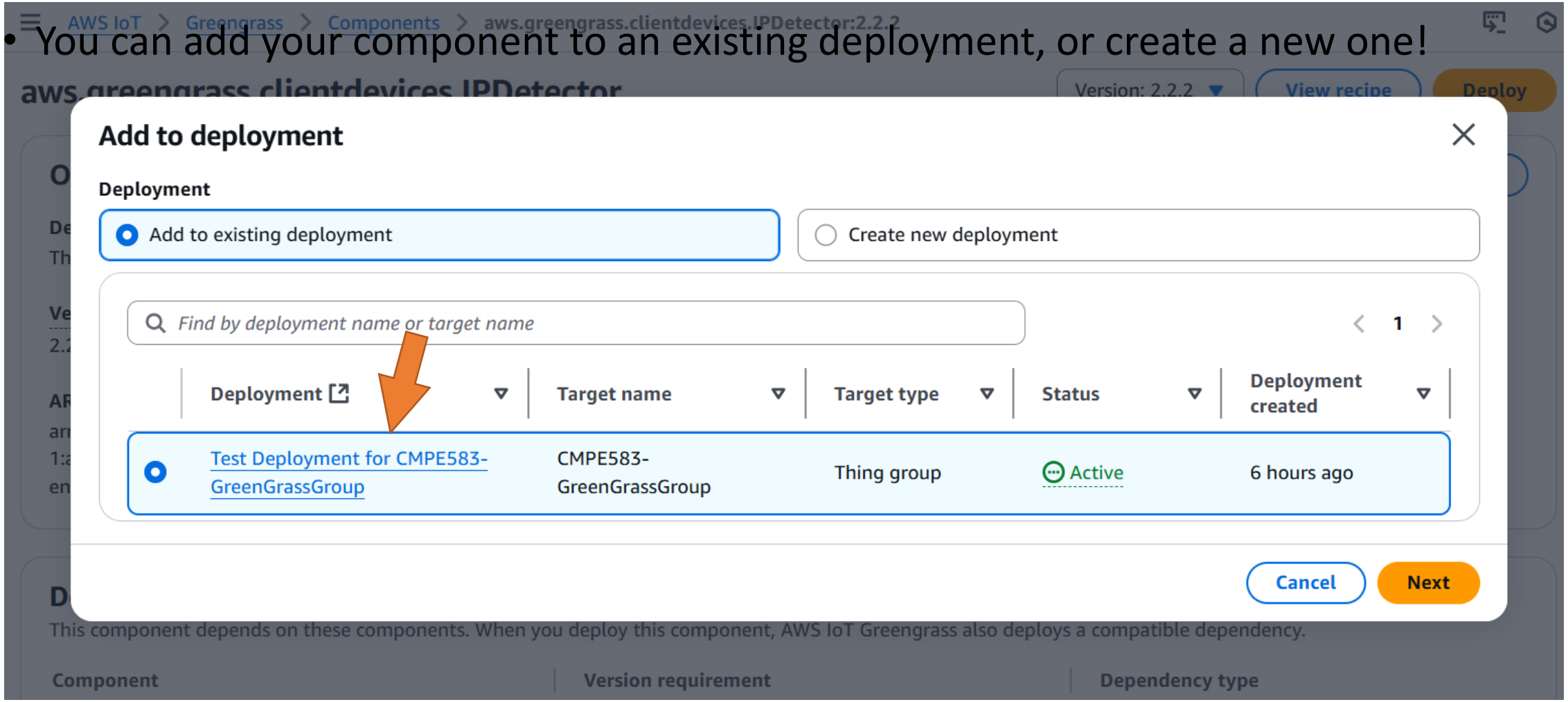
Deploy an IPDetector Component



Collecting IP Addresses of Core Devices

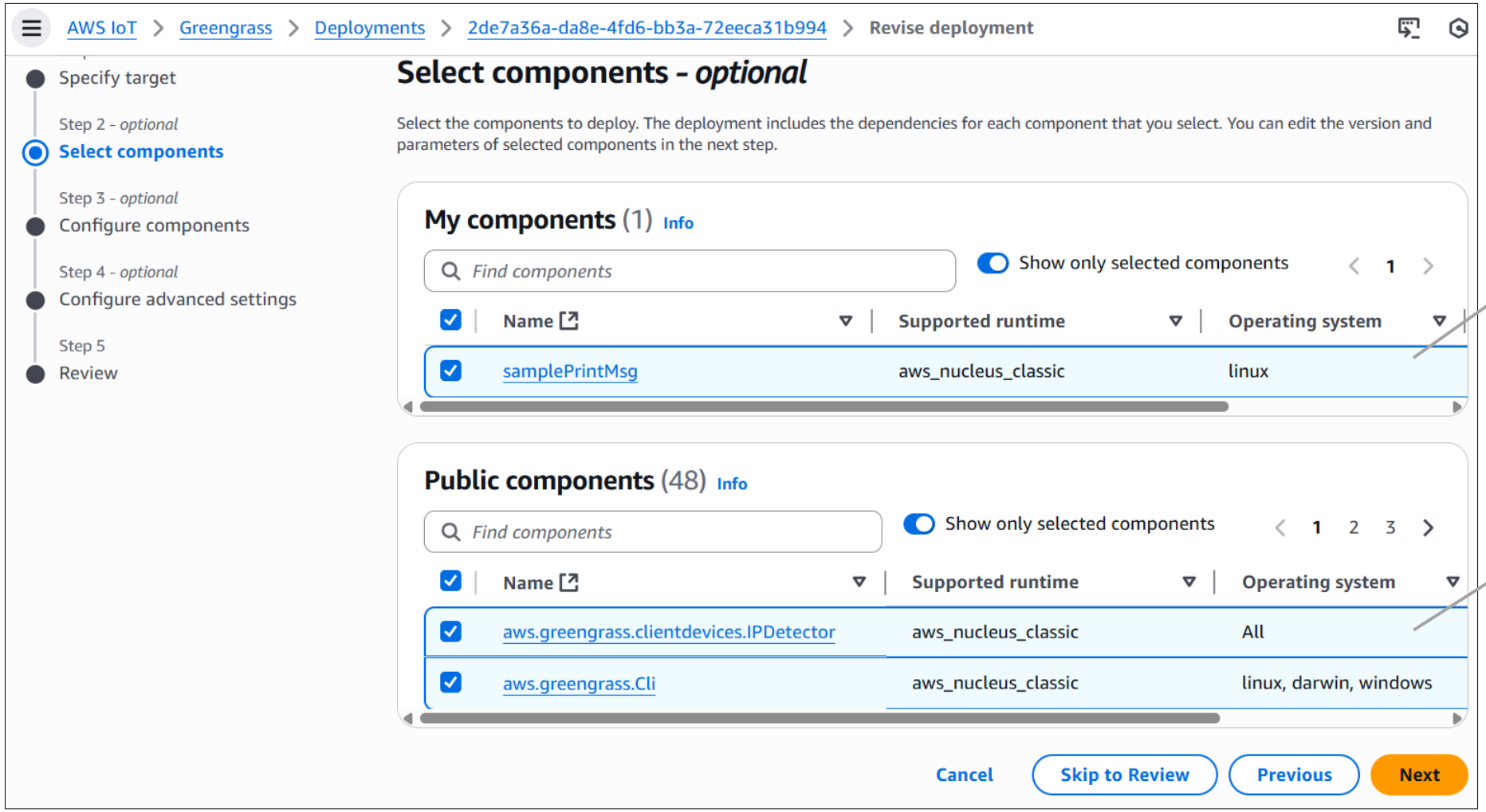
Add Public Component to Existing Deployment

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



Collecting IP Addresses of Core Devices

Select All Components You Want to Deploy



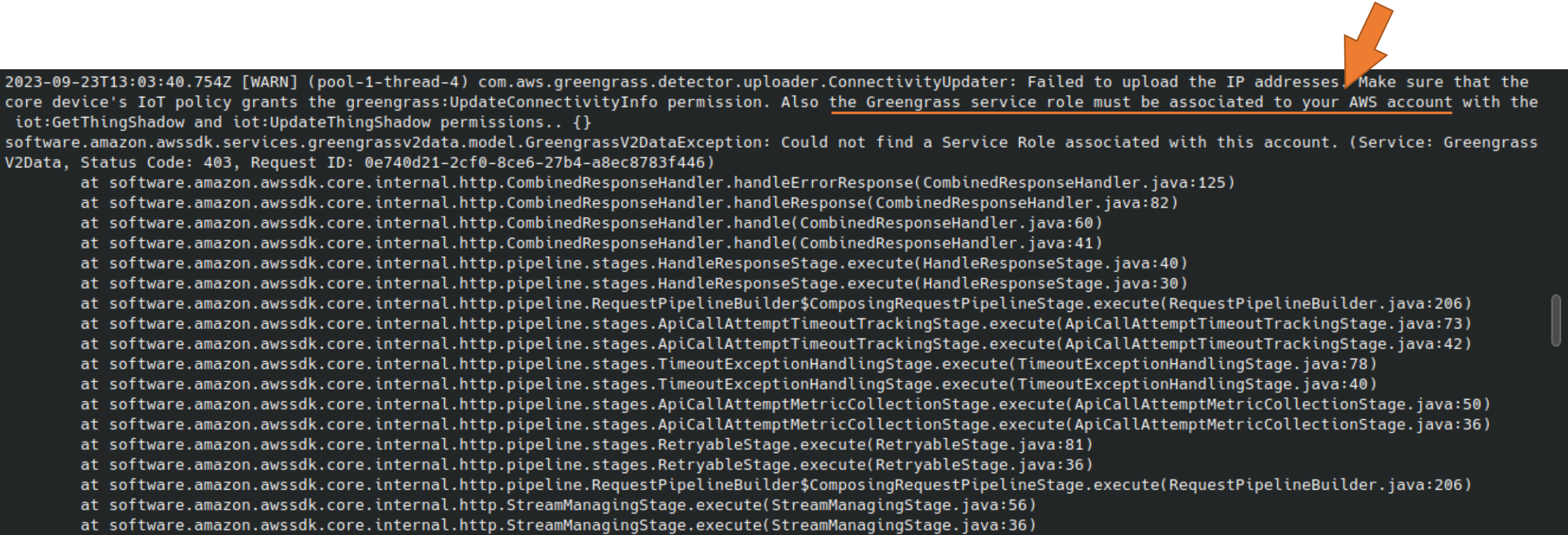
Custom component

AWS Provided component

Collecting IP Addresses of Core Devices

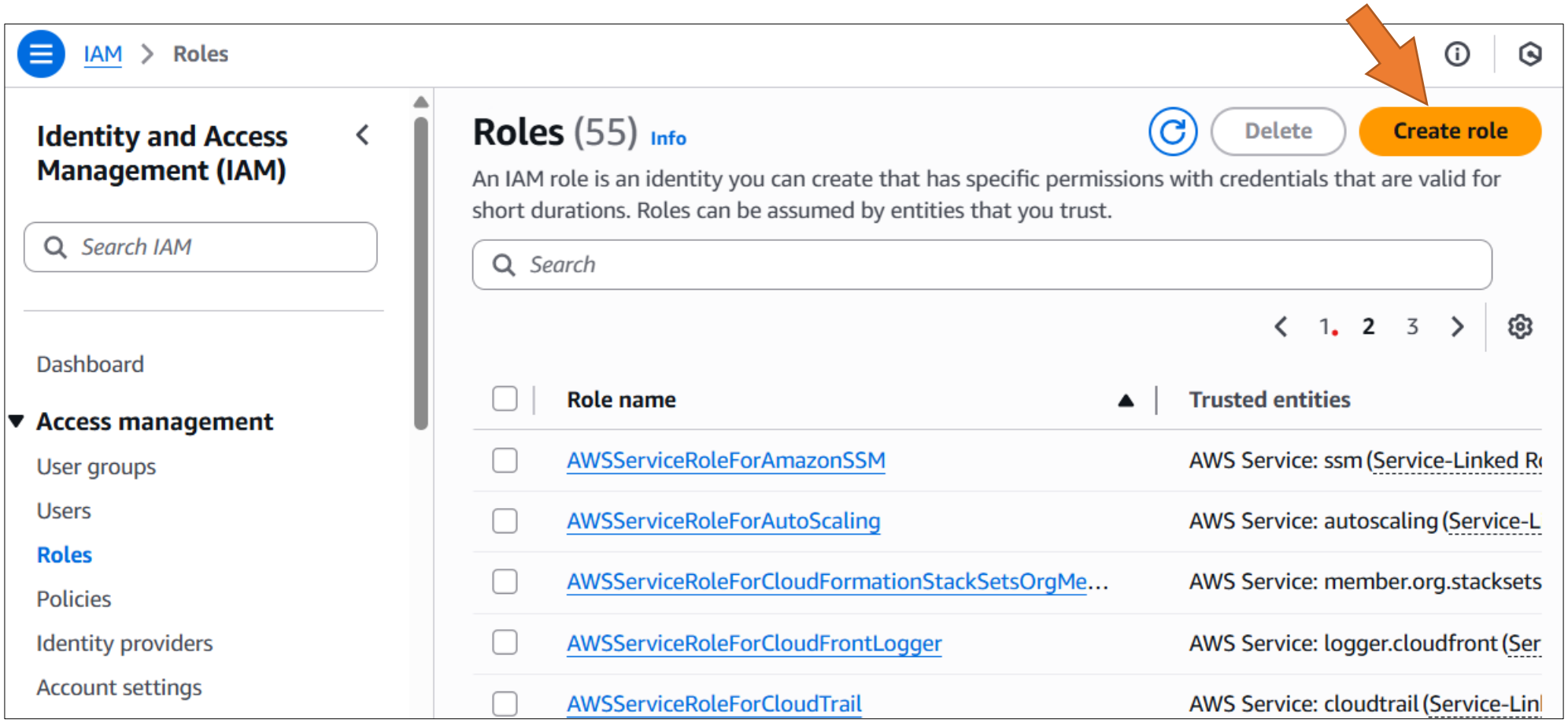
Troubleshooting

- 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.



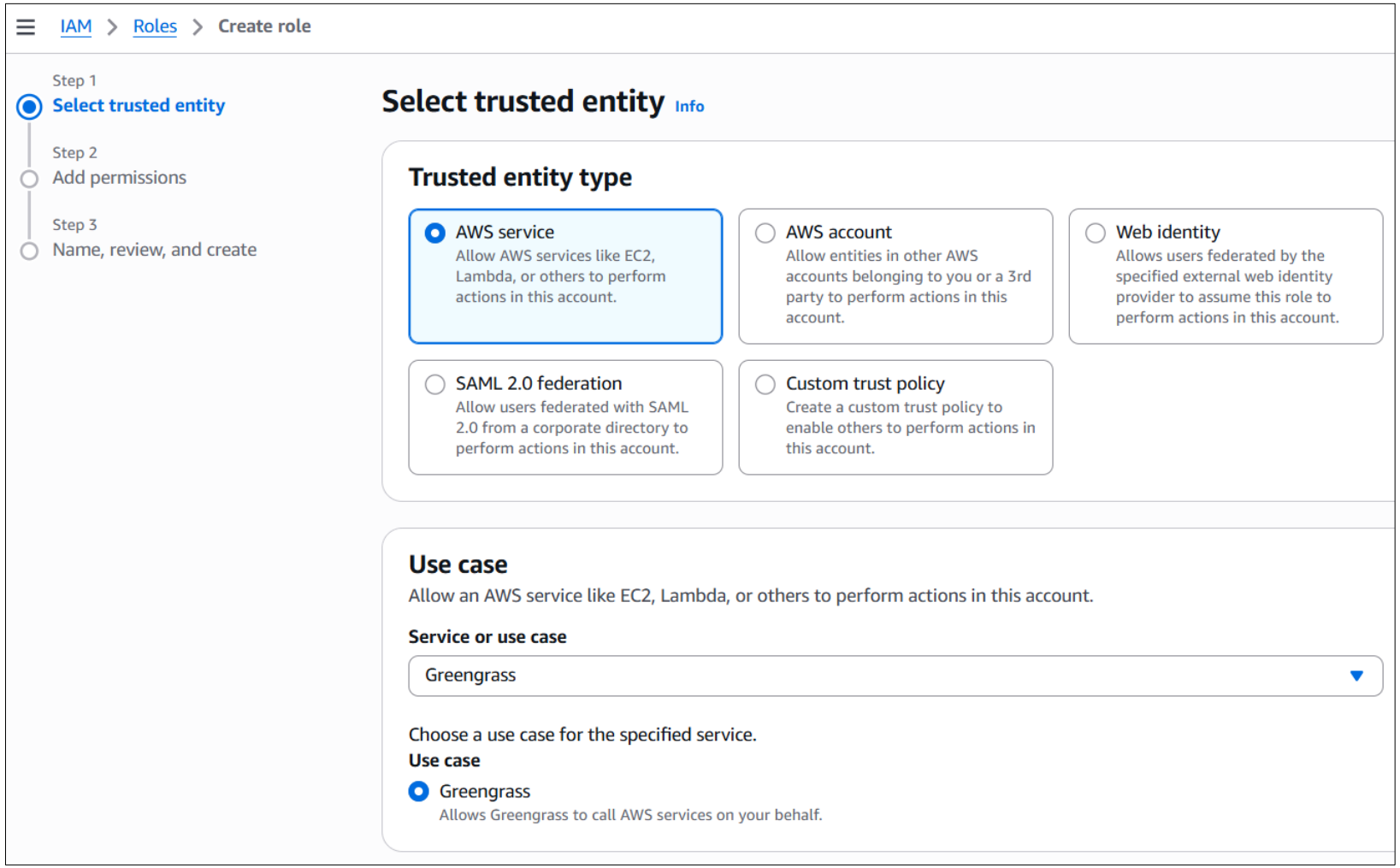
Troubleshooting

Create an IAM Role



Troubleshooting

Create an IAM Role

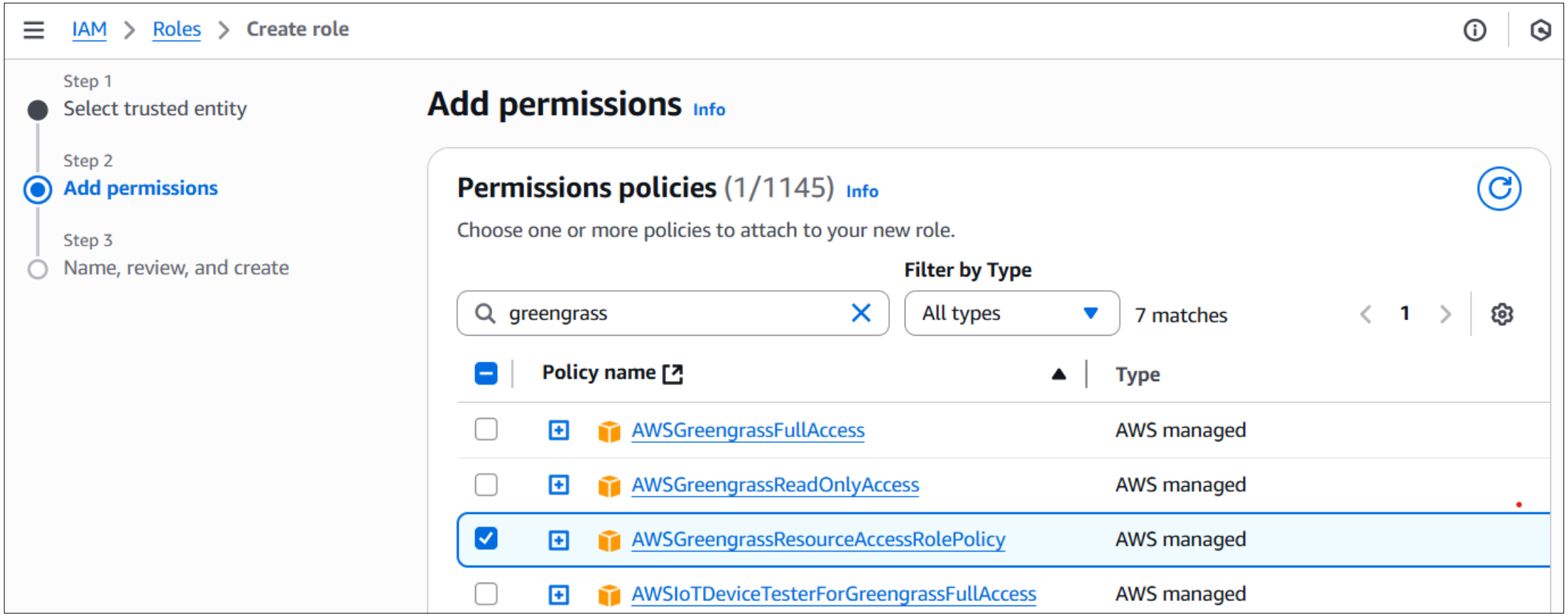


- Create a role that allows an AWS service to perform actions on your behalf.
- Allows Greengrass to call AWS services on your behalf.

Troubleshooting

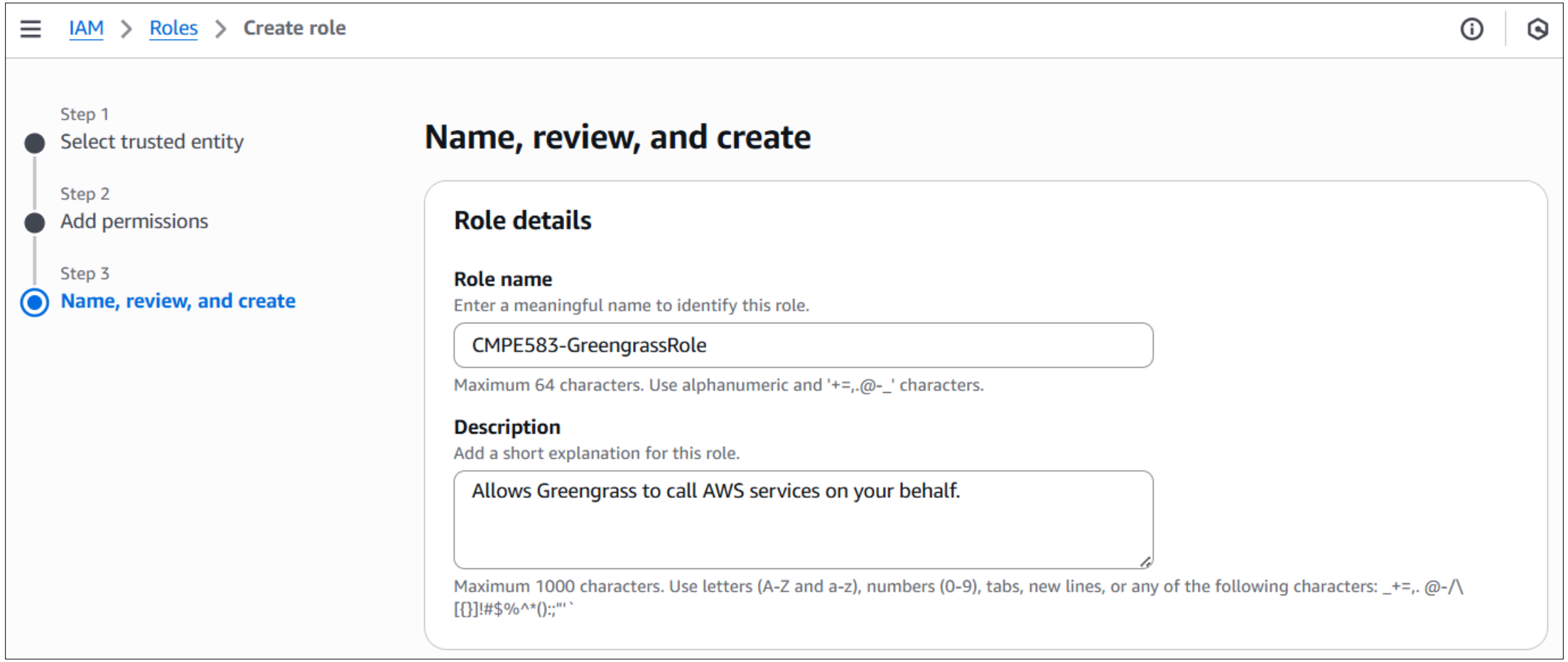
Add Permission to Created Role

- 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.



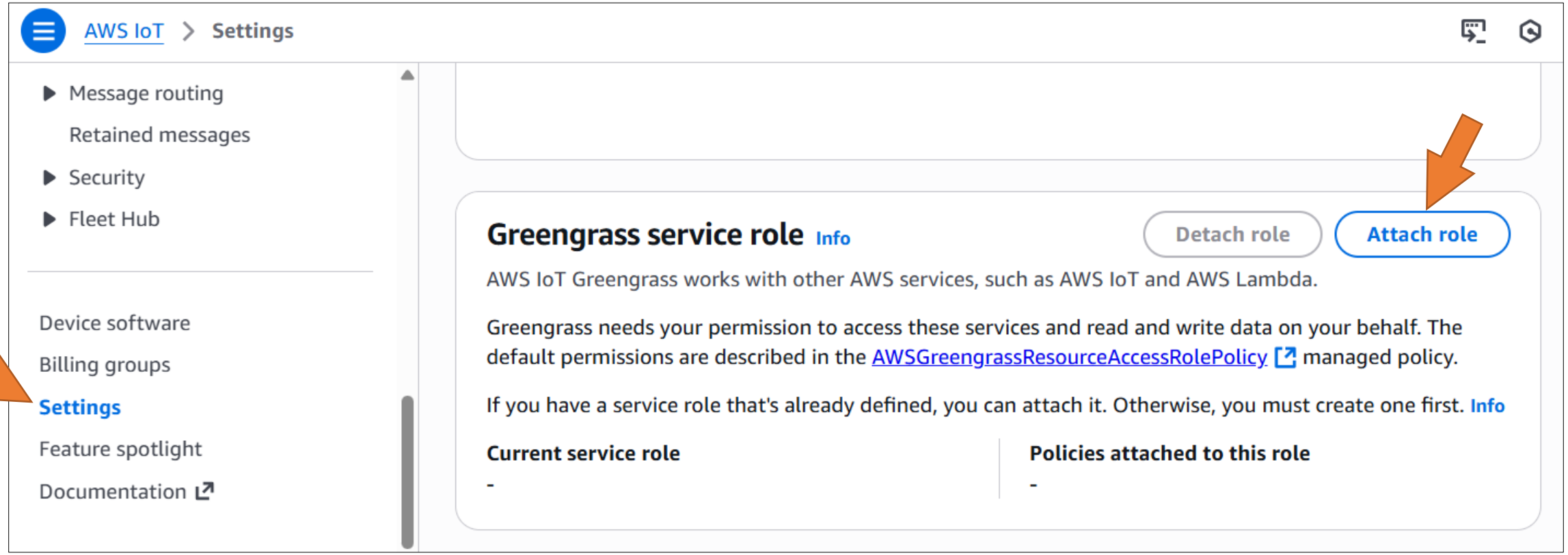
Troubleshooting

Review and Create Role



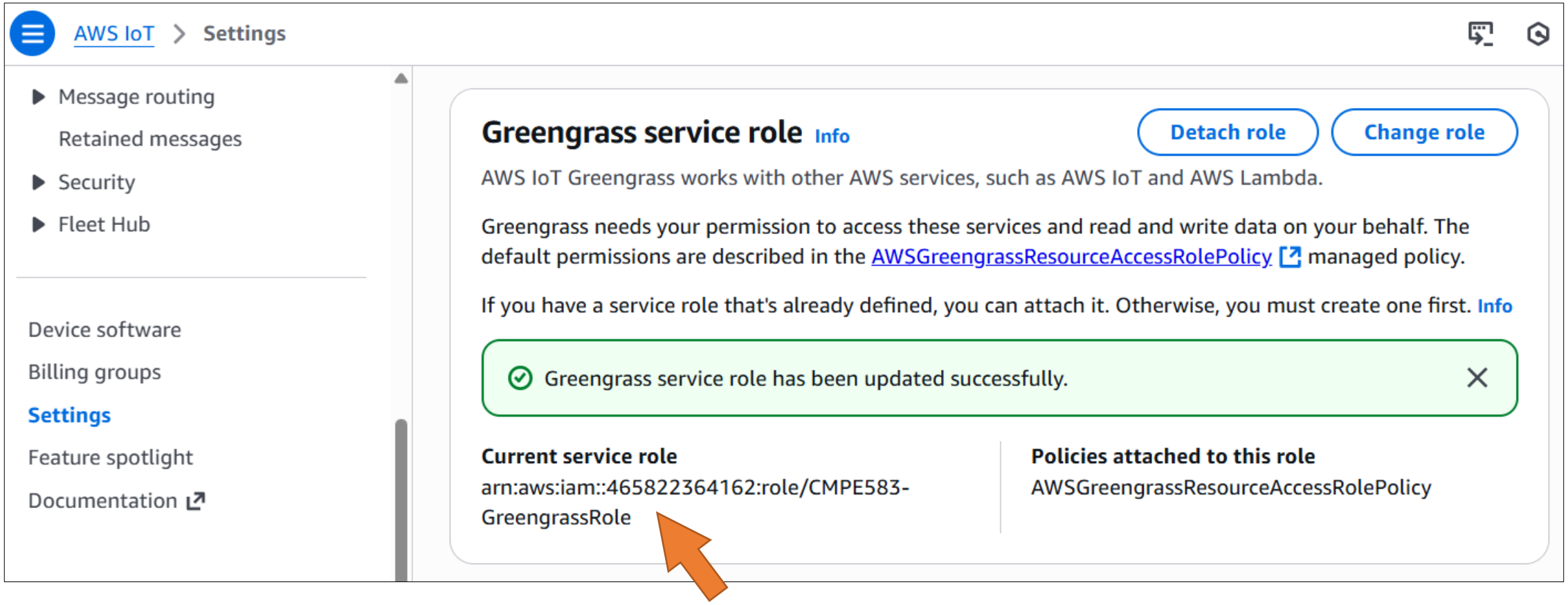
Troubleshooting

Attach Role to AWS IoT Service



Troubleshooting

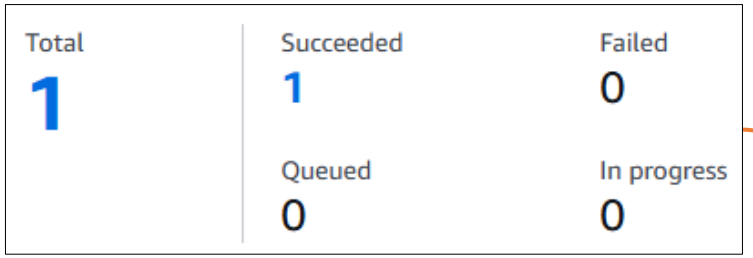
Attach Role to AWS IoT Service

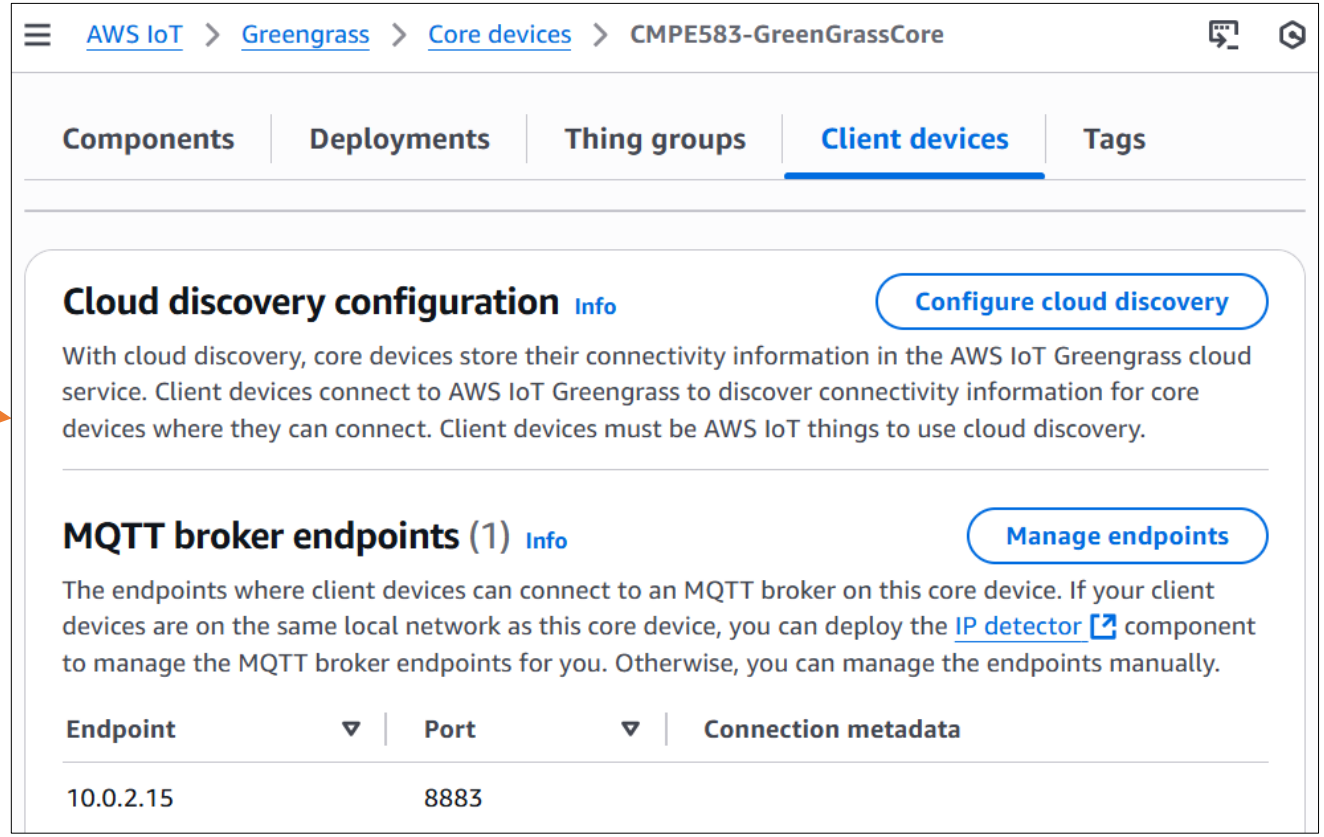


Troubleshooting

Verify the Deployment

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





Publish/Subscribe AWS IoT Core MQTT Messages

AWS IoT Greengrass Core IPC

- Components running on your core device can use the AWS IoT Greengrass Core interprocess communication (IPC) library in the AWS IoT Device SDK to communicate with the AWS IoT Greengrass nucleus and other Greengrass components.
- AWS IoT Greengrass provides an improved version of the IPC client: IPC client V2
- The IPC interface supports two types of operations:
  - **Request/response**
    - Components send a request to the IPC service and receive a response that contains the result of the request.
  - **Subscription**
    - Components send a subscription request to the IPC service and expect a stream of event messages in response.

AWS IoT Core MQTT Messaging IPC

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

MQTT

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

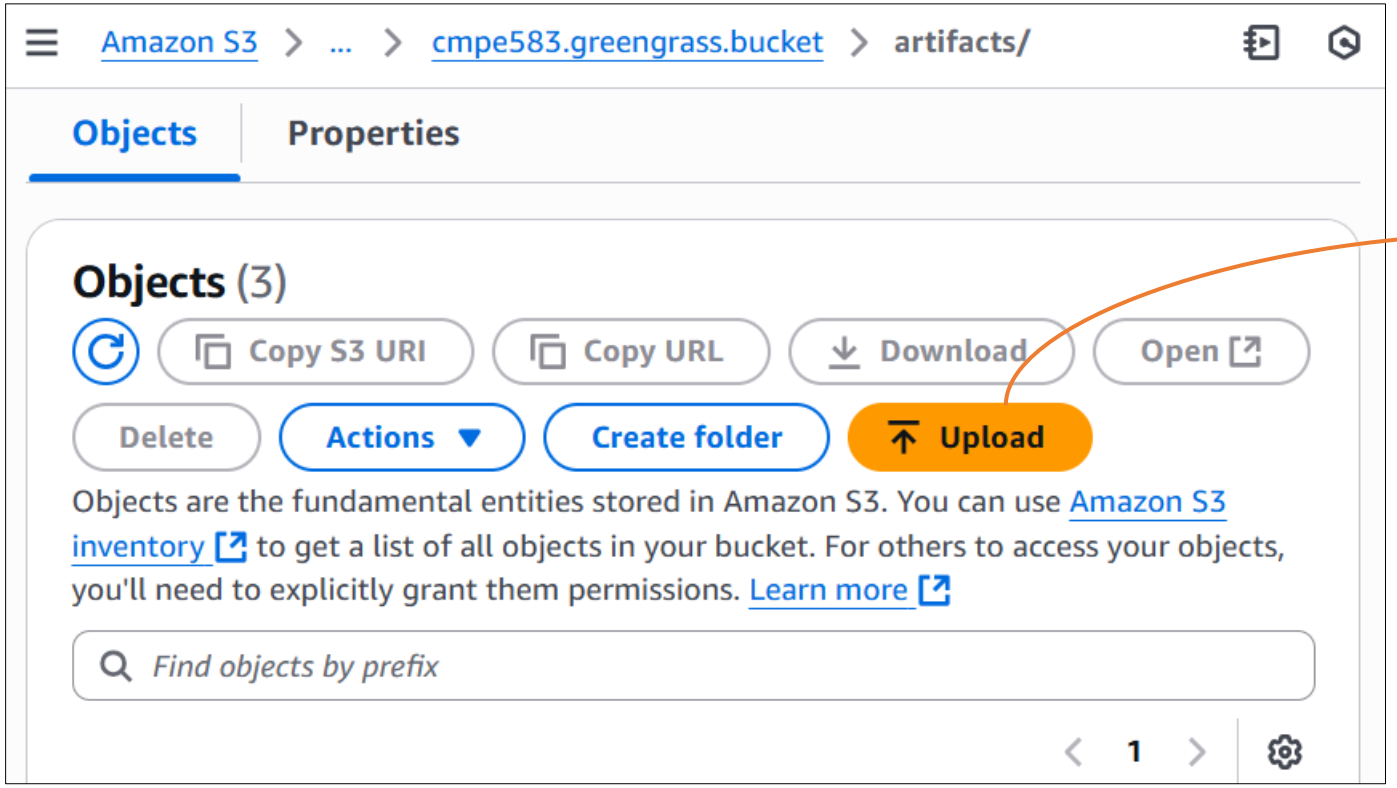
Steps for Deploying a Component Using Core IPC

1. Develop client code using Core MQTT IPC and upload artifacts to S3
2. Create the component and define its recipe
3. Set access control for MQTT topics and deploy the component
4. Verify and test the component using AWS MQTT Client

Step 1: Develop & Upload Client-Side Code

Via AWS IoT Greengrass (Console)

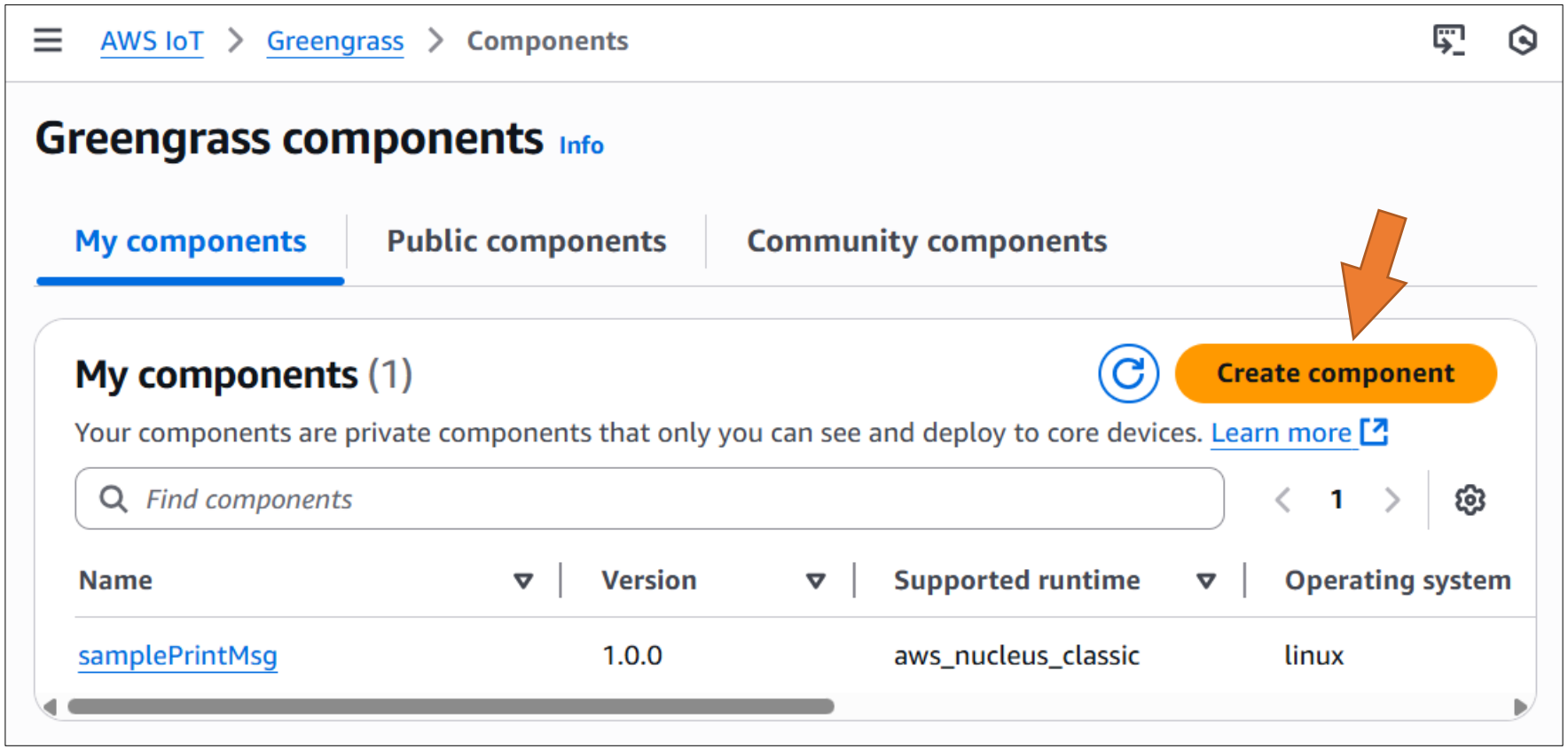
- Upload your artifacts to a folder in your S3 bucket




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

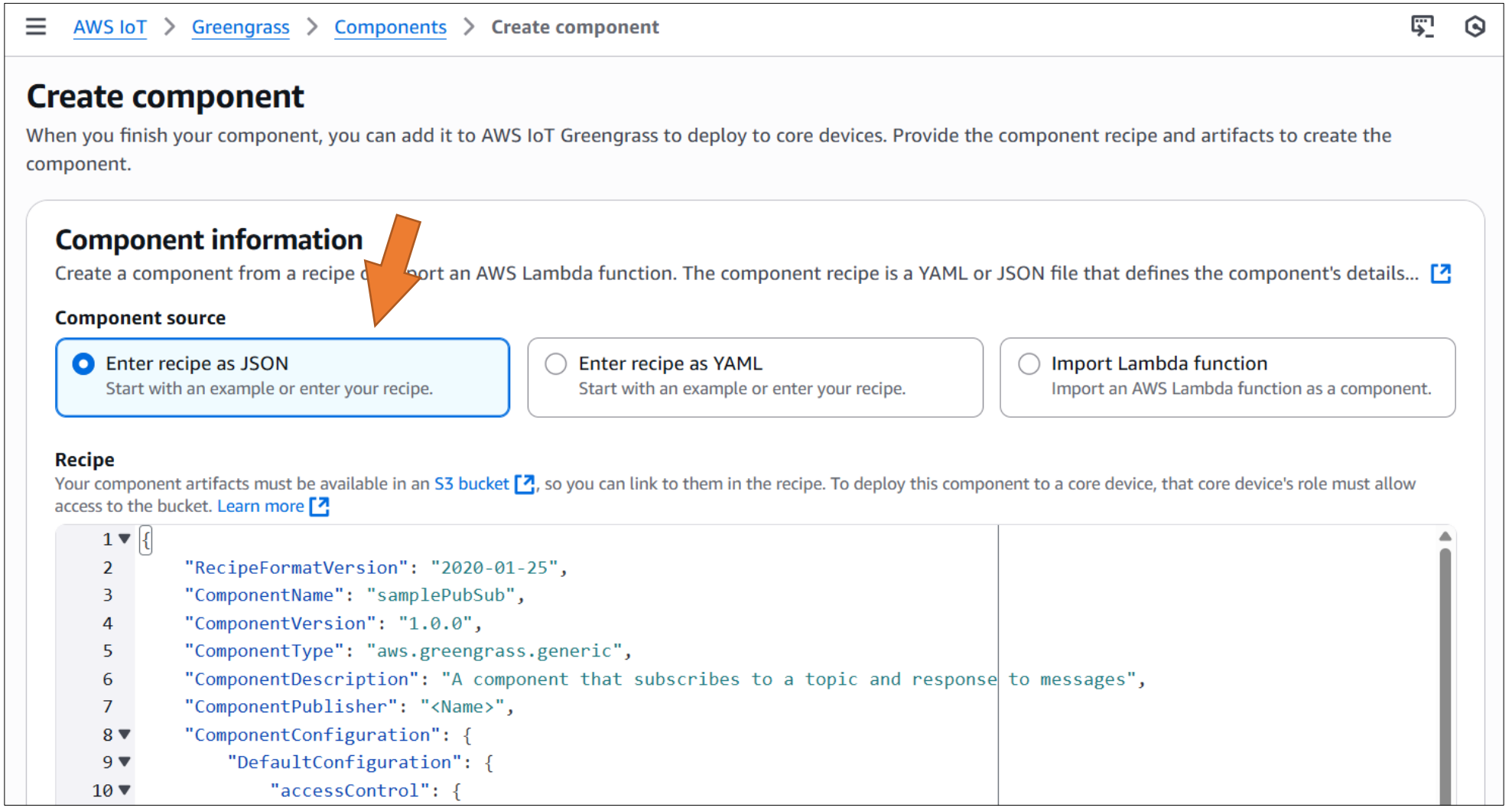
Step 2: Create a Custom Component

Via AWS IoT Greengrass (Console)



Step 2: Create a Custom Component

Enter the Recipe



Step 2: Create a Custom Component

Adjust the Access Control

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

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

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

Step 2: Create a Custom Component

Adjust the Access Control

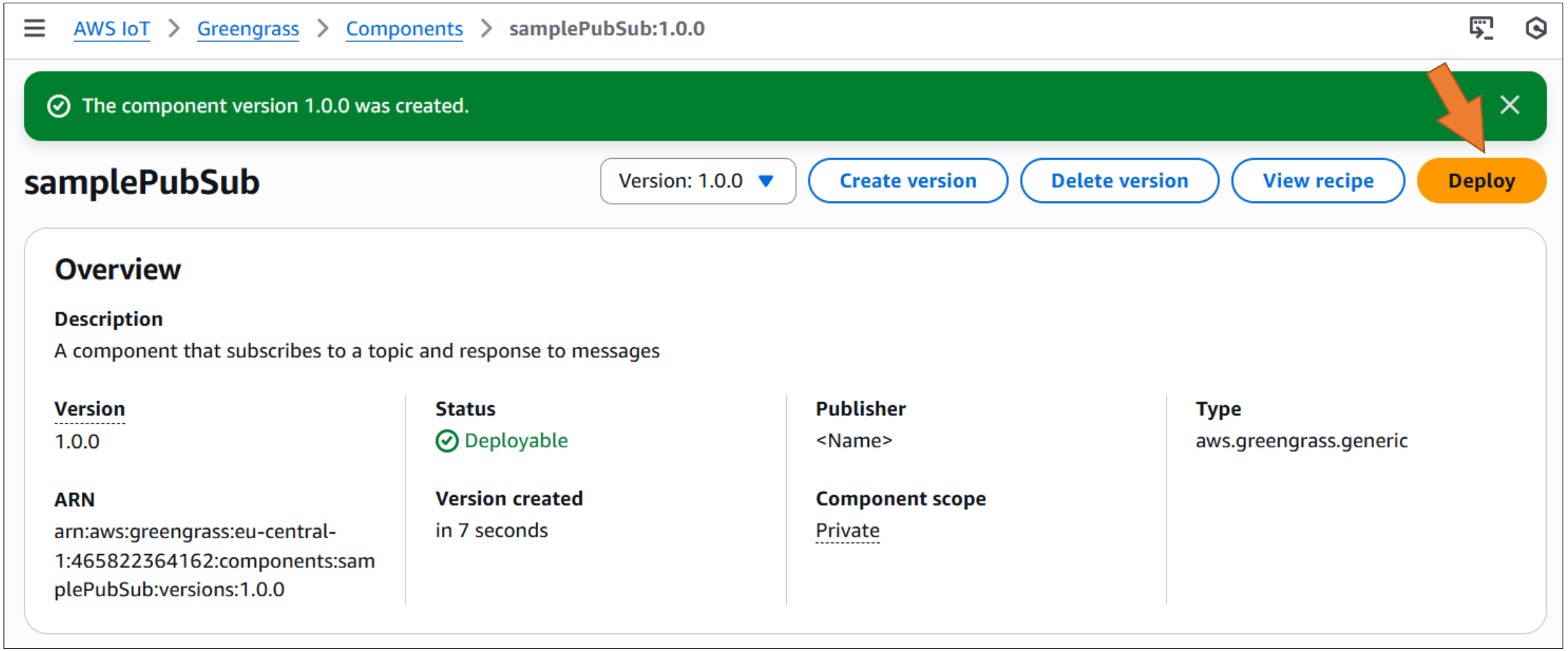
```
"Manifests": [
    {
        "Lifecycle": {
            "Install": "pip3 install --user awsiotsdk botocore --break-system-packages",
            "Run": "python3 -u {artifacts:path}/cmpe583-PubSub.py cmpe/request cmpe/response\n"
        },
        "Artifacts": [
            {
                "Uri":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

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

awsiotsdk and botocore libraries are required.

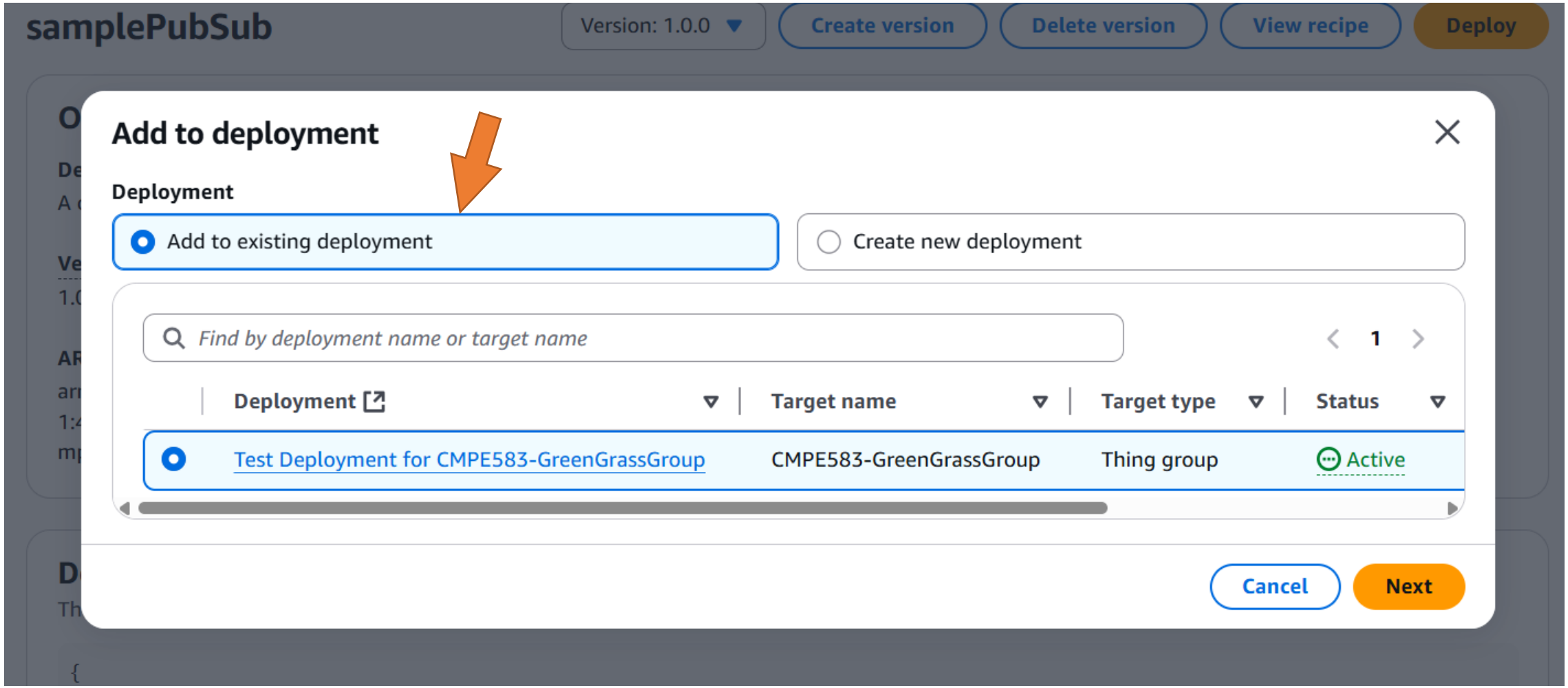
Step 3: Deploy Your Component

Via AWS IoT Greengrass (Console)



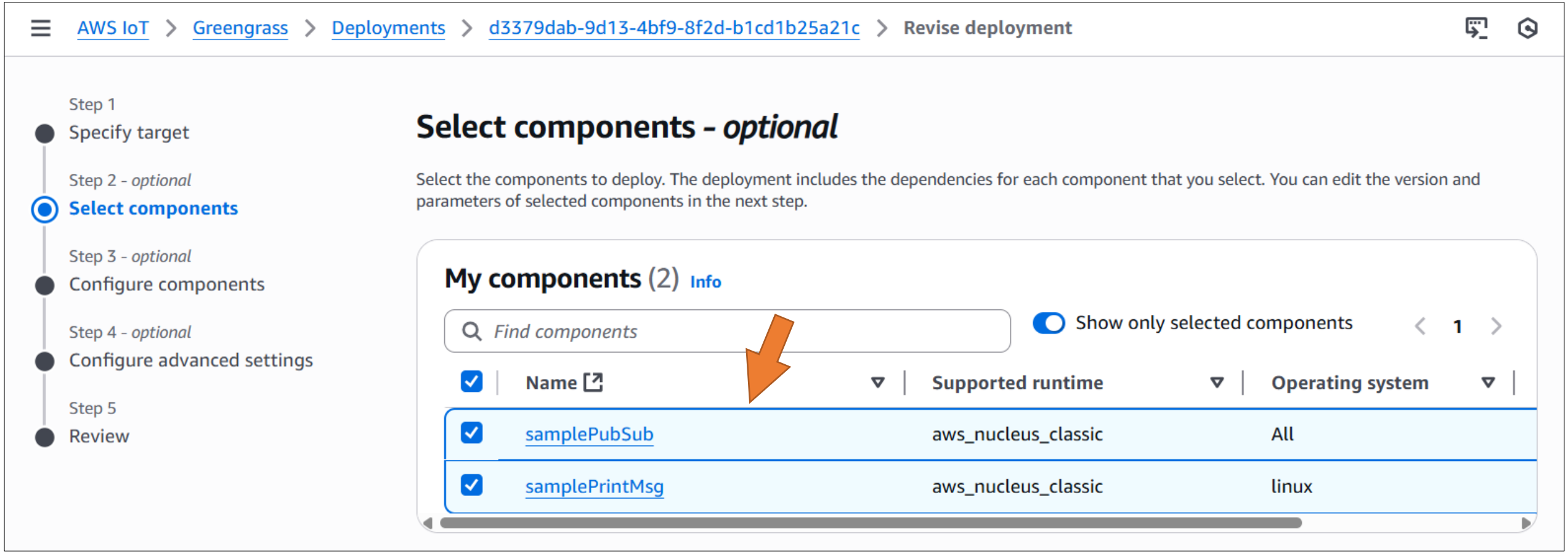
Step 3: Deploy Your Component

Add Custom Component to Existing Deployment



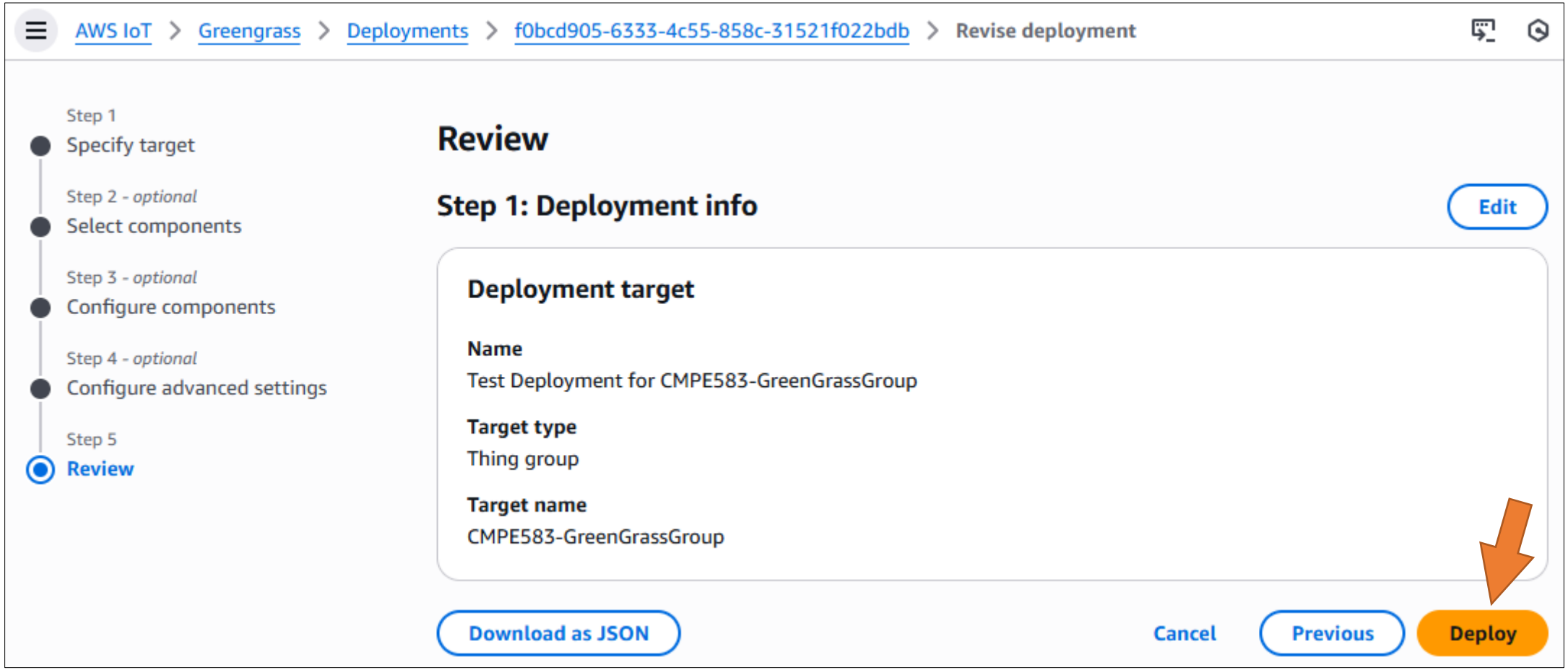
Step 3: Deploy Your Component

Select the Component



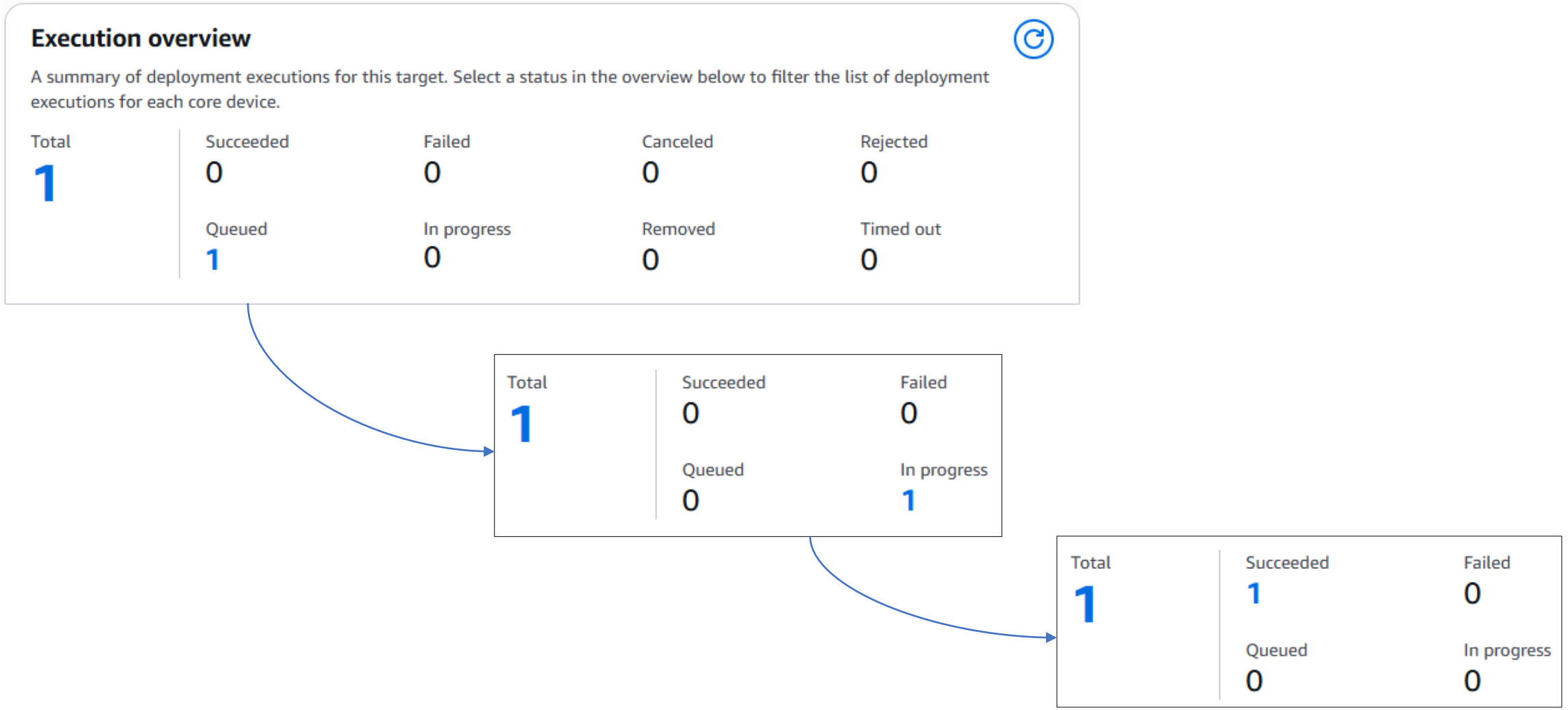
Step 3: Deploy the Component

Review & Deploy



Step 4: Verify the Component

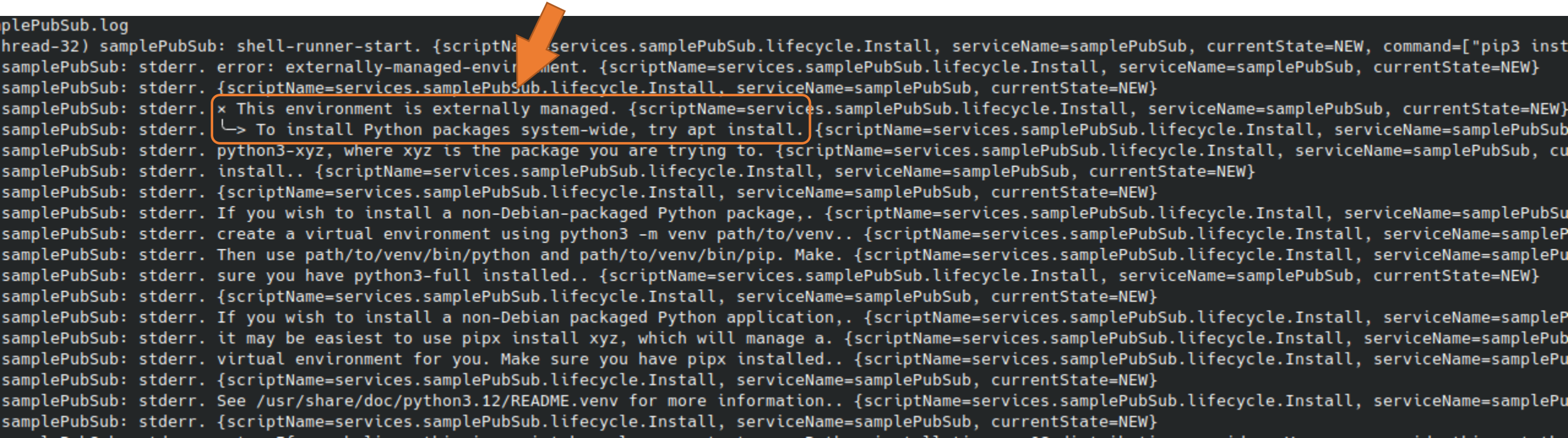
Check AWS IoT Greengrass Console



Troubleshooting

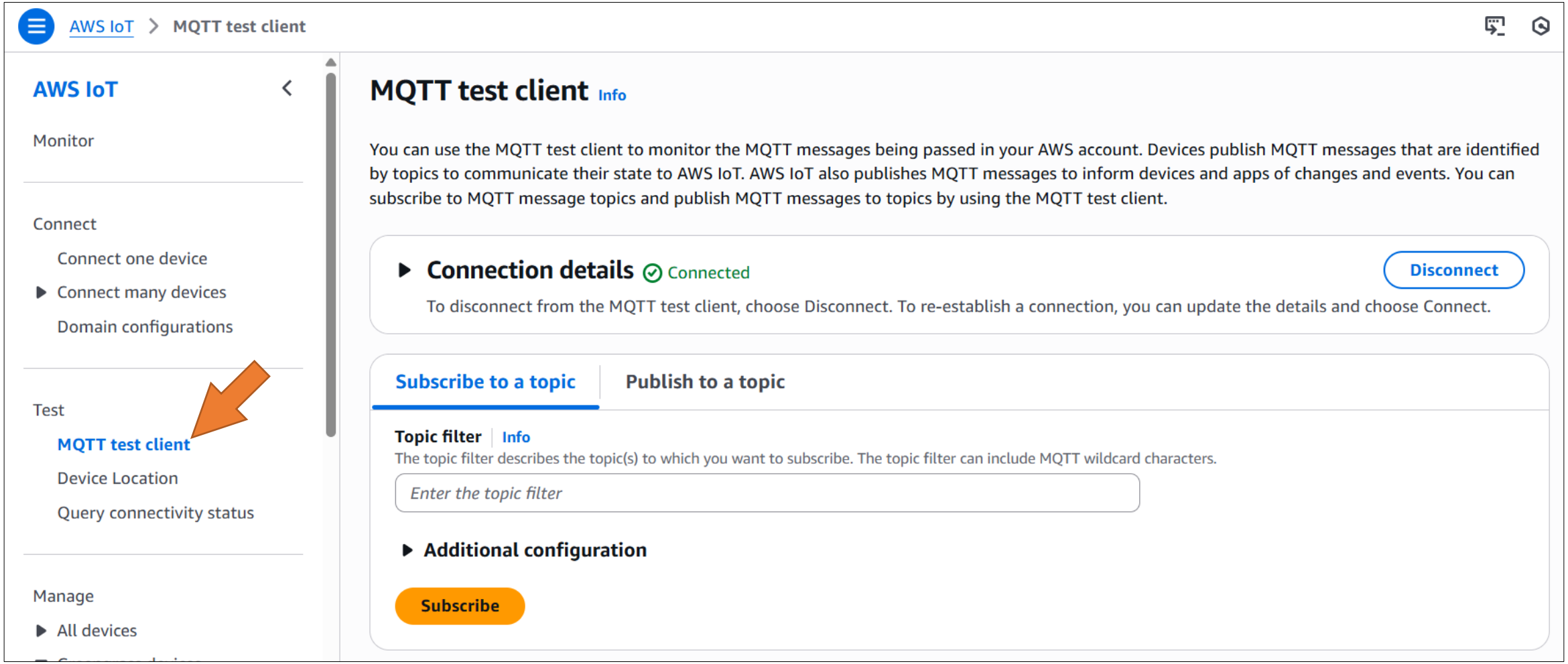
Check greengrass.log File

- 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.



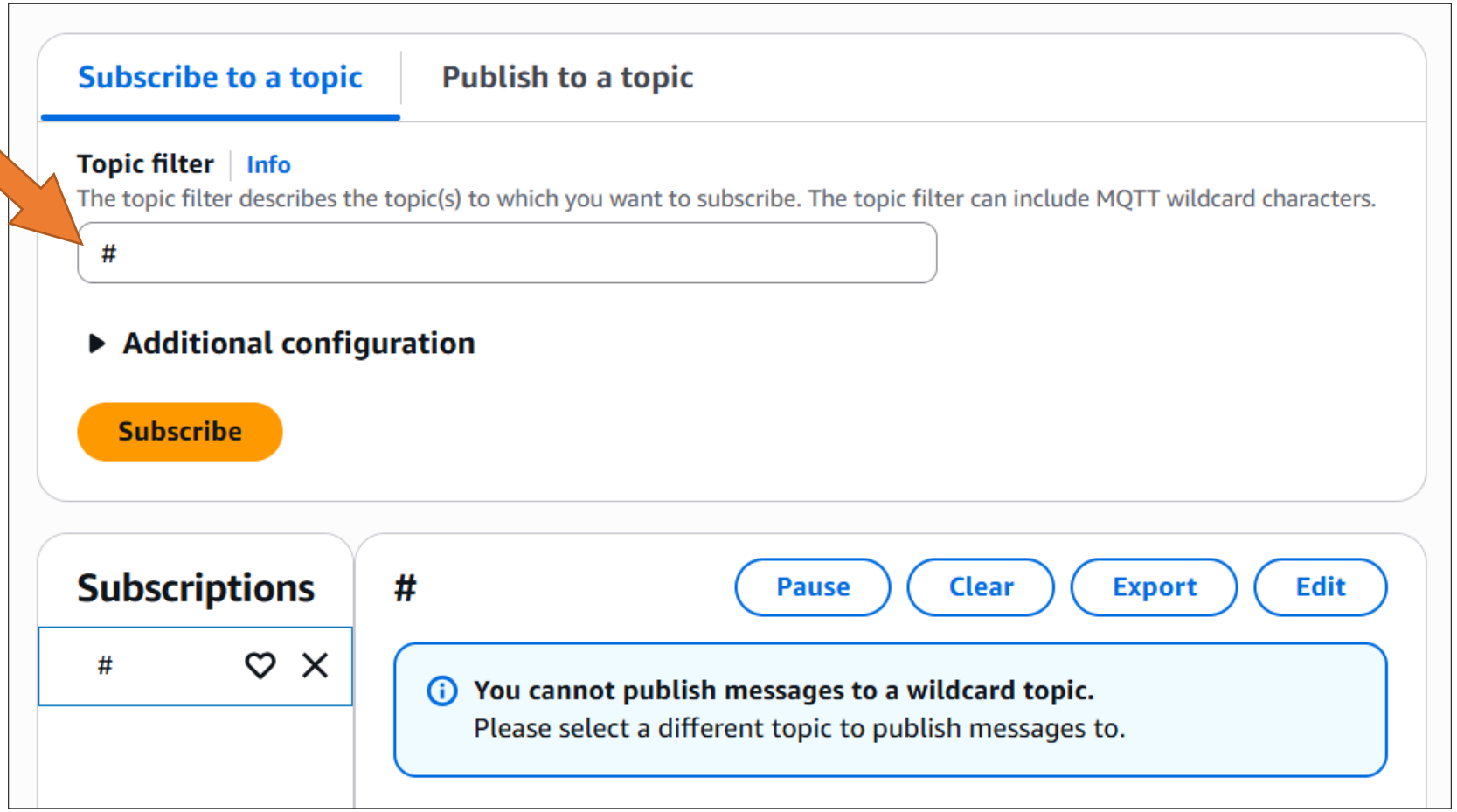
Test Your Deployment

Via AWS MQTT Test Client



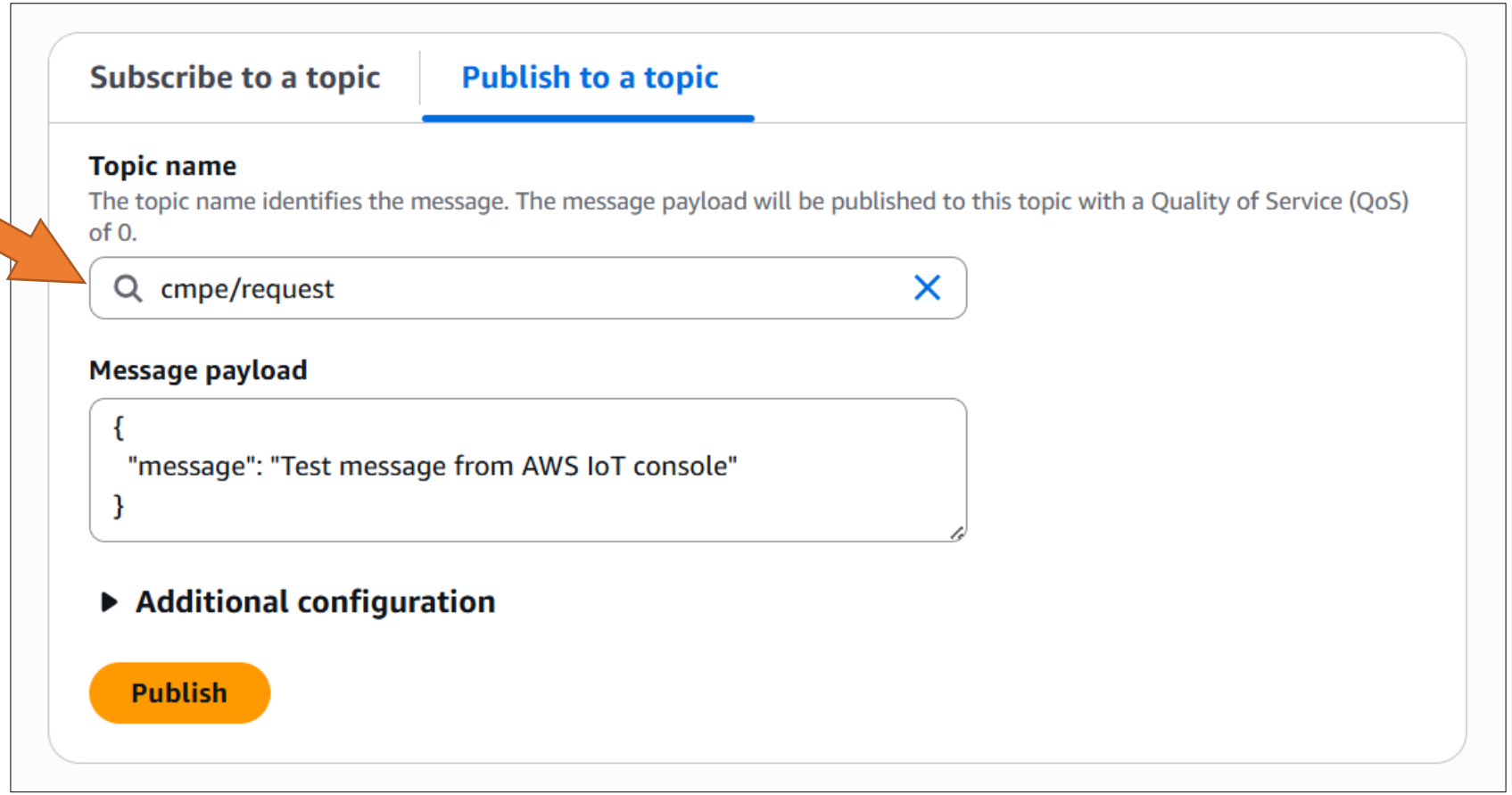
Test Your Deployment

Subscribe All Topics



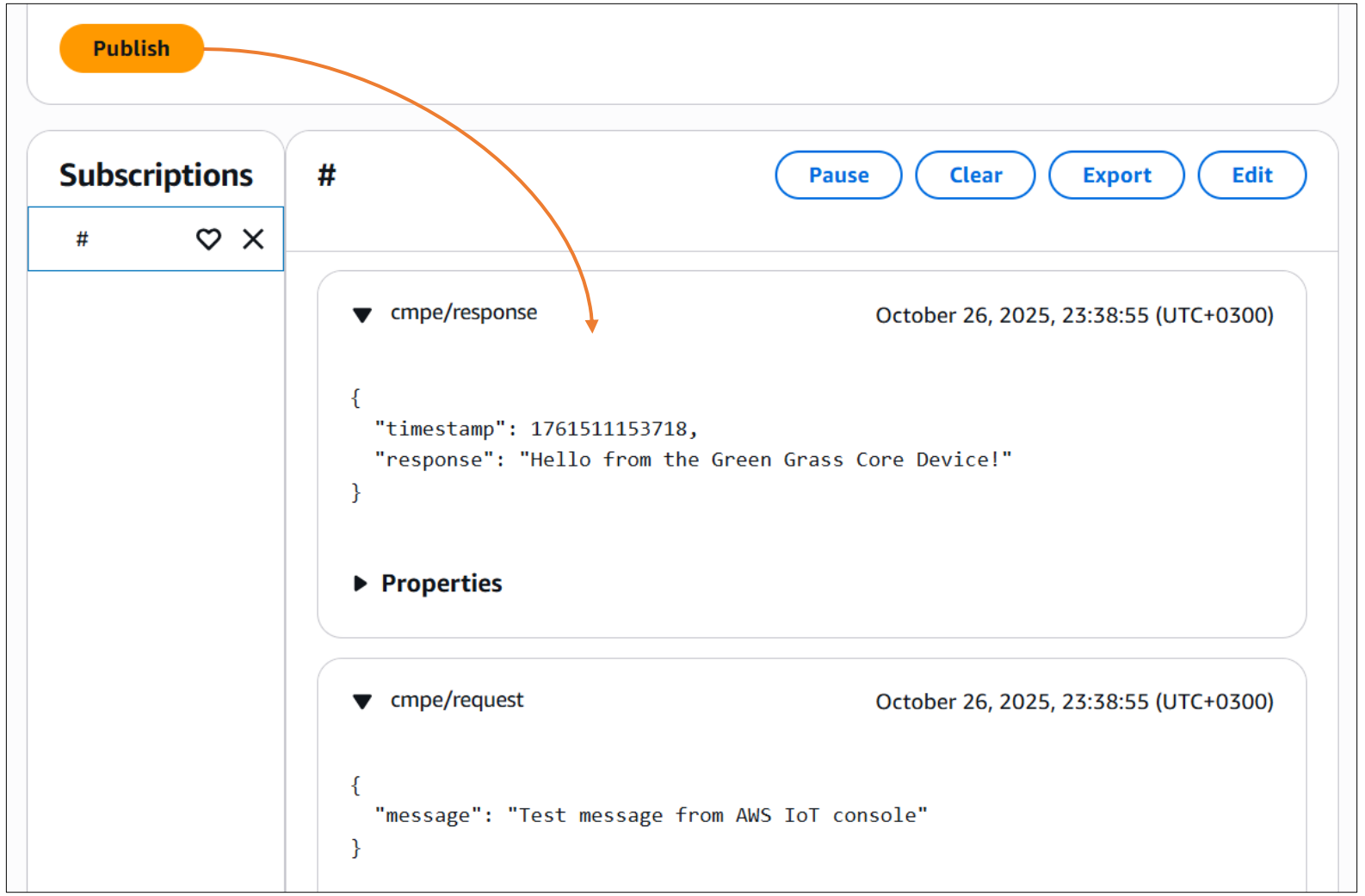
Test Your Deployment

Send a Test Message



Test Your Deployment

Check If the Client Responds Back



Other Use-Cases of AWS IoT Greengrass Core IPC

- Interact with component lifecycle (component-to-component or component-to-cloud).
  - Pause component, resume component
  - Listen to lifecycle events (start, stop, etc.)
- Interact with component configuration.
  - Get and set component configuration parameters
- Retrieve secret values (access AWS Secrets Manager secrets securely).
  - Gets the value of a secret that you store on the core device
- Authenticate and authorize client devices.
  - Verify the identity of a client device
  - Validates a client device's credentials
  - Verify whether a client device has permission to perform an action on a resource

AWS Lambda Functions on Core Devices

Run AWS Lambda Functions on Core Devices

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

Lambda Function Requirements

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

Lambda Function Lifecycle

On-demand functions

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

Long-lived functions

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

On-demand vs Long-lived Functions

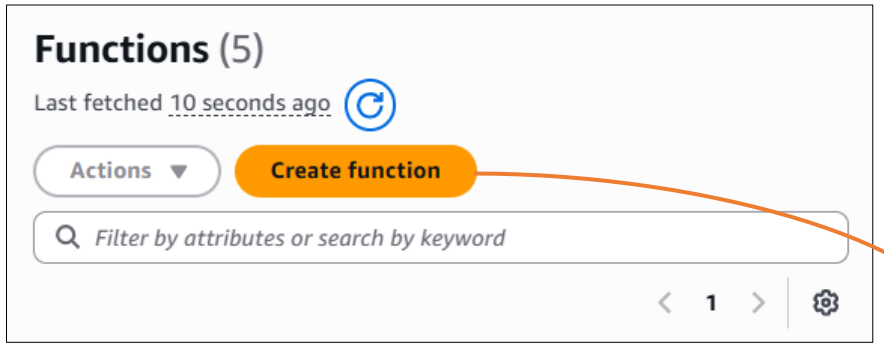
Feature	On-demand Function	Long-lived (Pinned) Function
Lifecycle Trigger	Starts only when invoked by an event (e.g., an MQTT message or an IPC call).	Starts automatically with the Greengrass Core component and runs continuously.
Execution Duration	**Short-lived.** The function's container is shut down after a period of inactivity.	**Persistent.** Runs indefinitely. If the function fails, Greengrass automatically restarts it.
State Management	Primarily **Stateless**. Cannot reliably maintain state in memory between invocations.	Primarily **Stateful**. Ideal for maintaining state, open connections, and managing in-memory caches.
Resource Usage	**Low / Intermittent**. Consumes CPU and memory only when active, making it efficient for resource-constrained devices.	**High / Constant**. Consumes resources continuously as long as the Greengrass Core is running.
Configuration	Configured with **Pinned: false** (default).	Configured with **Pinned: true**.
Ideal Use Cases	• Responding to sensor data triggers • Message transformation and routing • Simple, event-driven logic that runs infrequently	• Continuous data acquisition from local devices • Running a local ML inference model. • Managing a persistent connection to a local database or device.

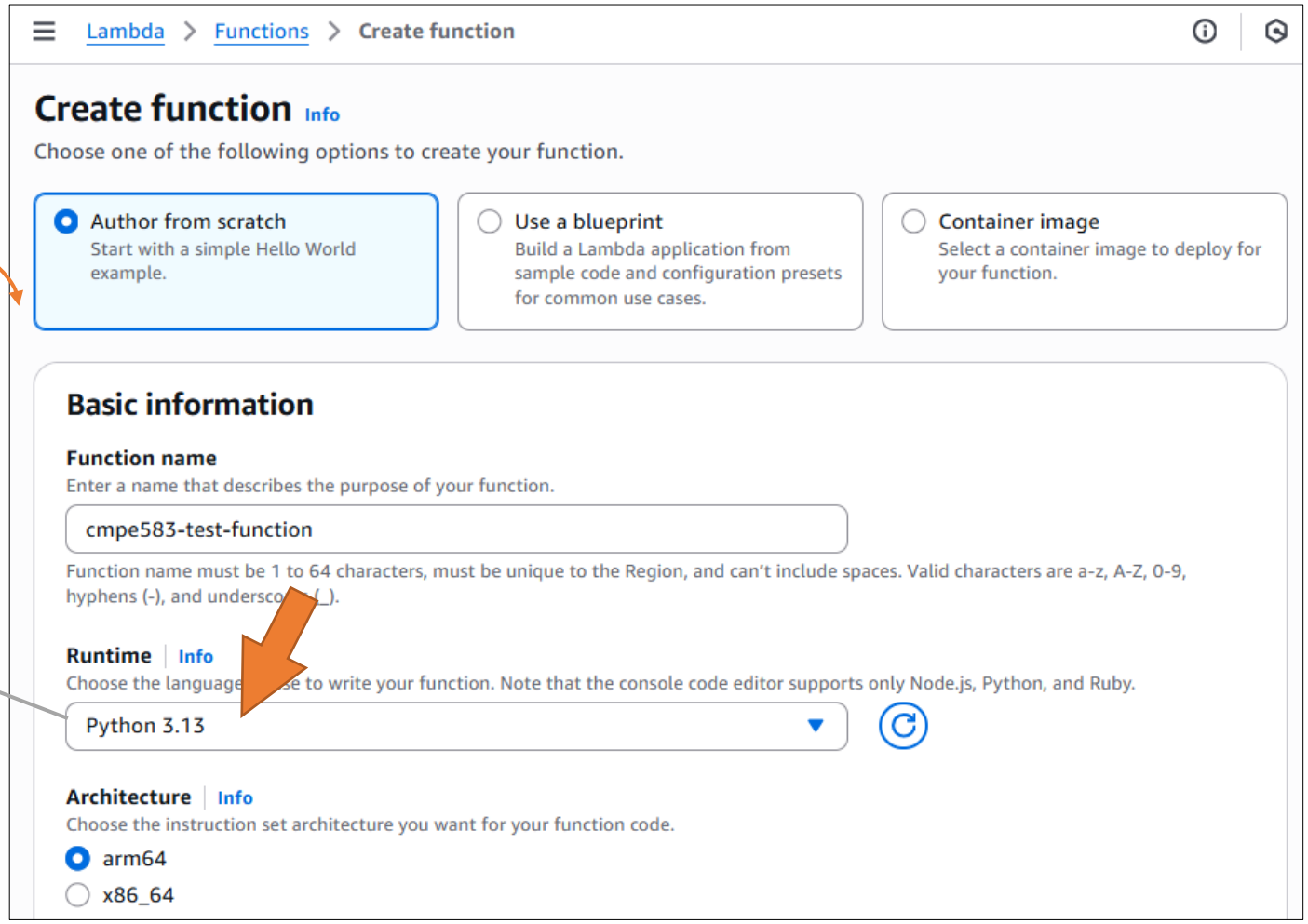
Steps for Deploying a Component Using Lambda

1. Create an AWS Lambda function and publish it
2. Create a custom component using the Lambda function
3. Deploy your Greengrass component and define a trigger function
4. Verify and test the Greengrass component

Step 1: Create AWS Lambda Function

Via AWS Lambda (Console)





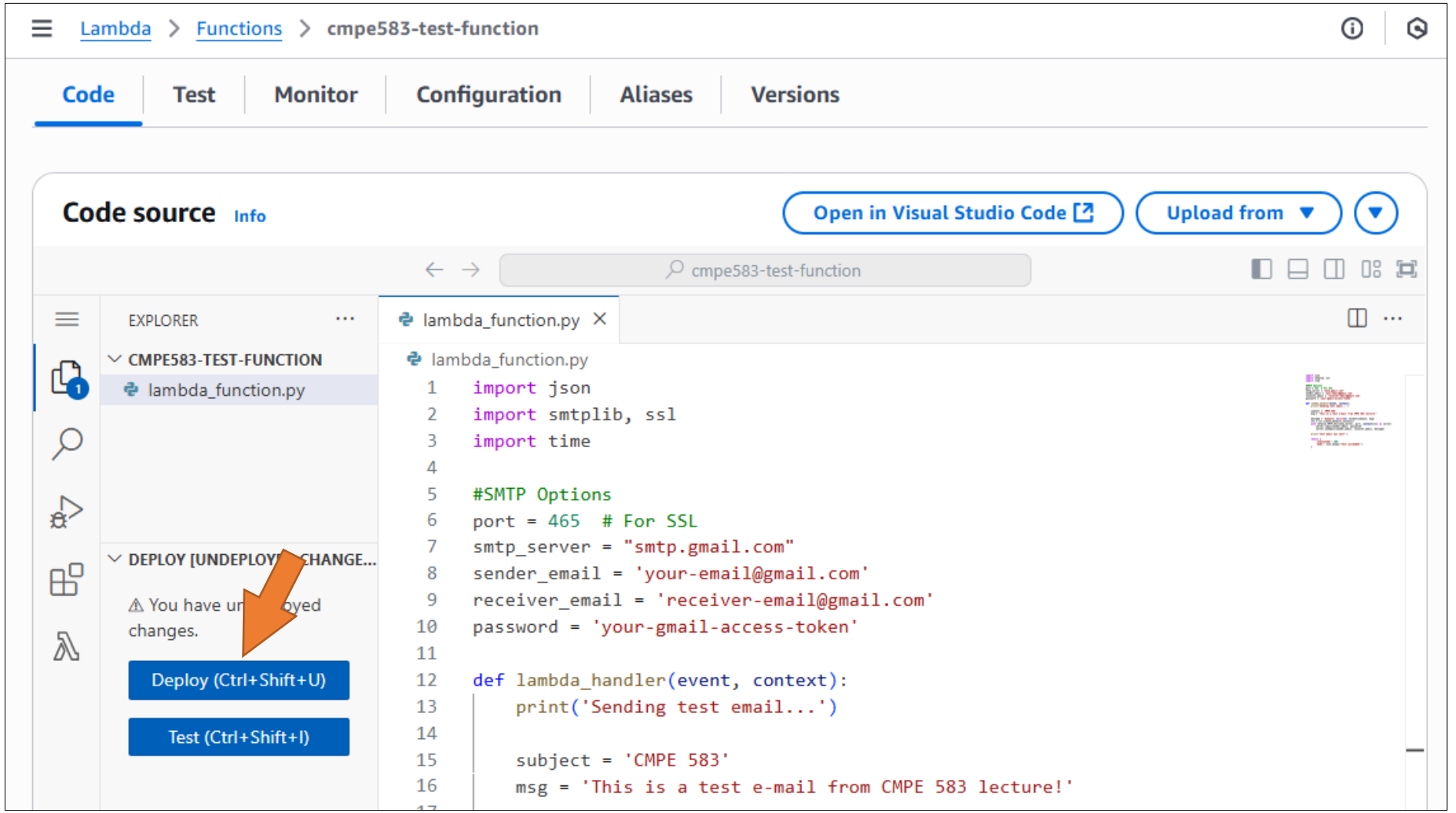
🚨 **Potential Issues:**

Runtime Version Mismatch

Try to use same Runtime and Architecture on AWS and your Core Device! Otherwise use FunctionRuntimeOverride in your Greengrass configuration

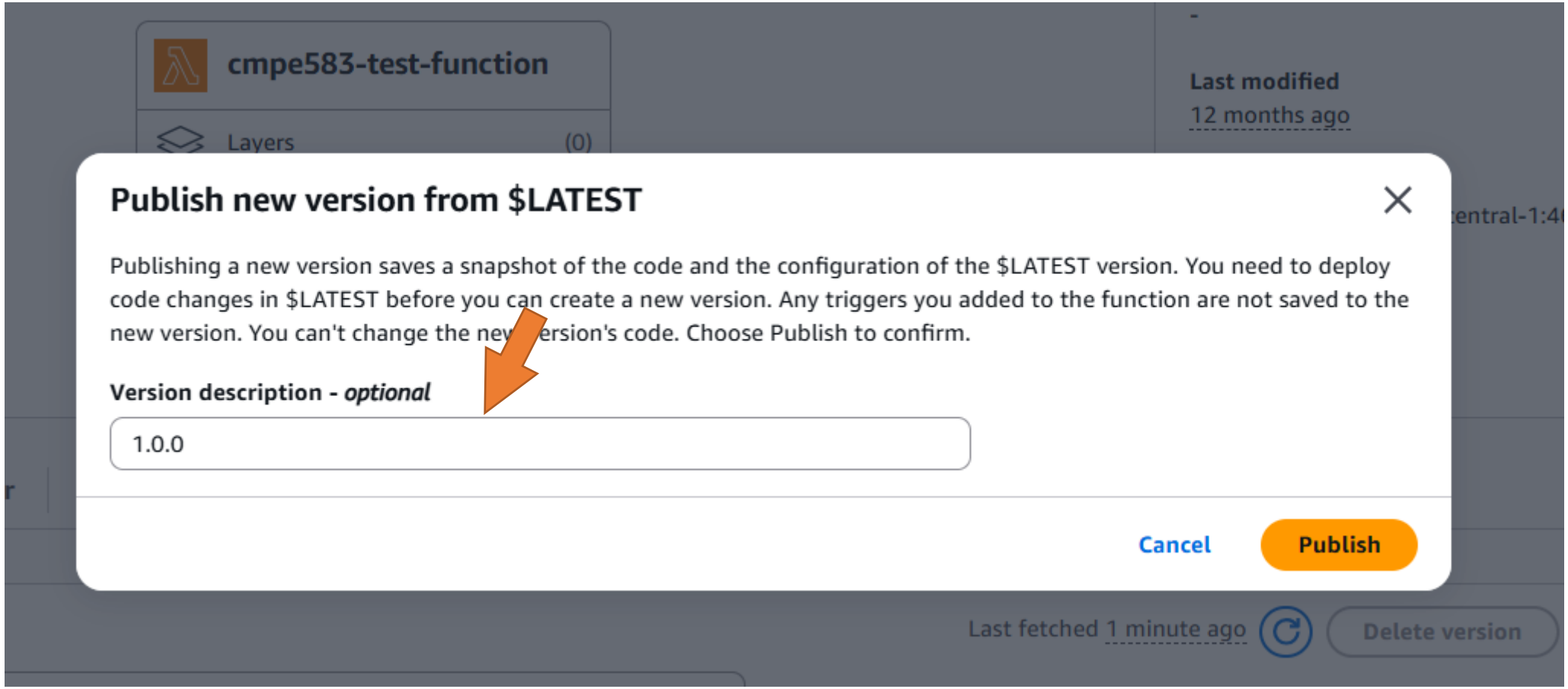
Step 1: Create AWS Lambda Function

Deploy the Lambda Function



Step 1: Create AWS Lambda Function

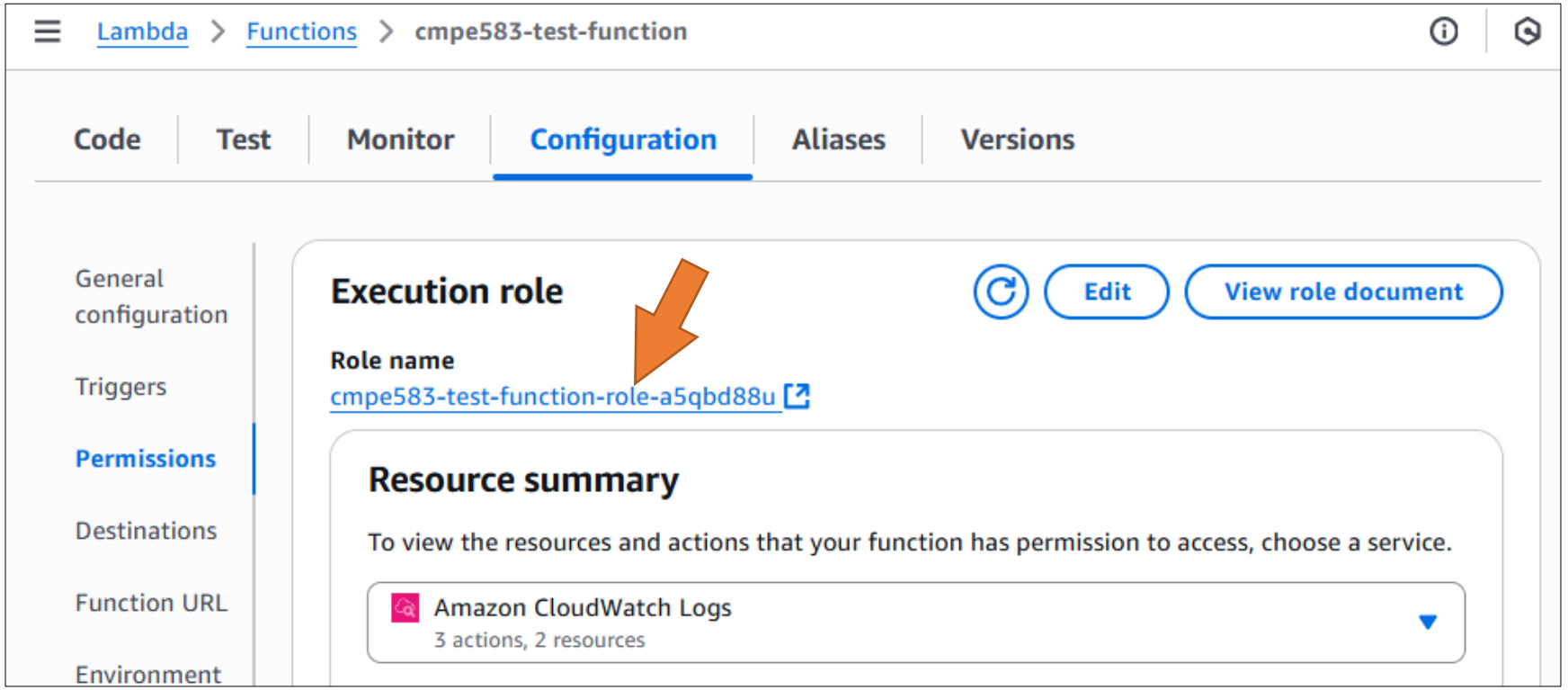
Publish the First Version



Step 1: Create AWS Lambda Function

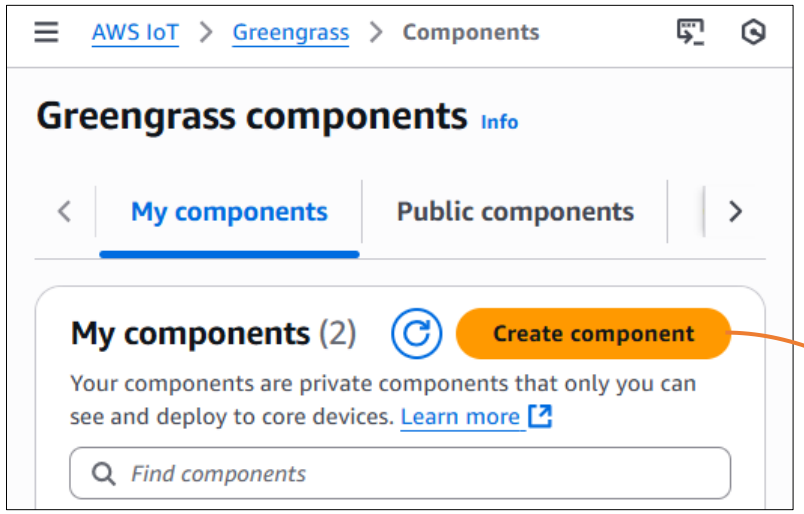
Configure Lambda Role

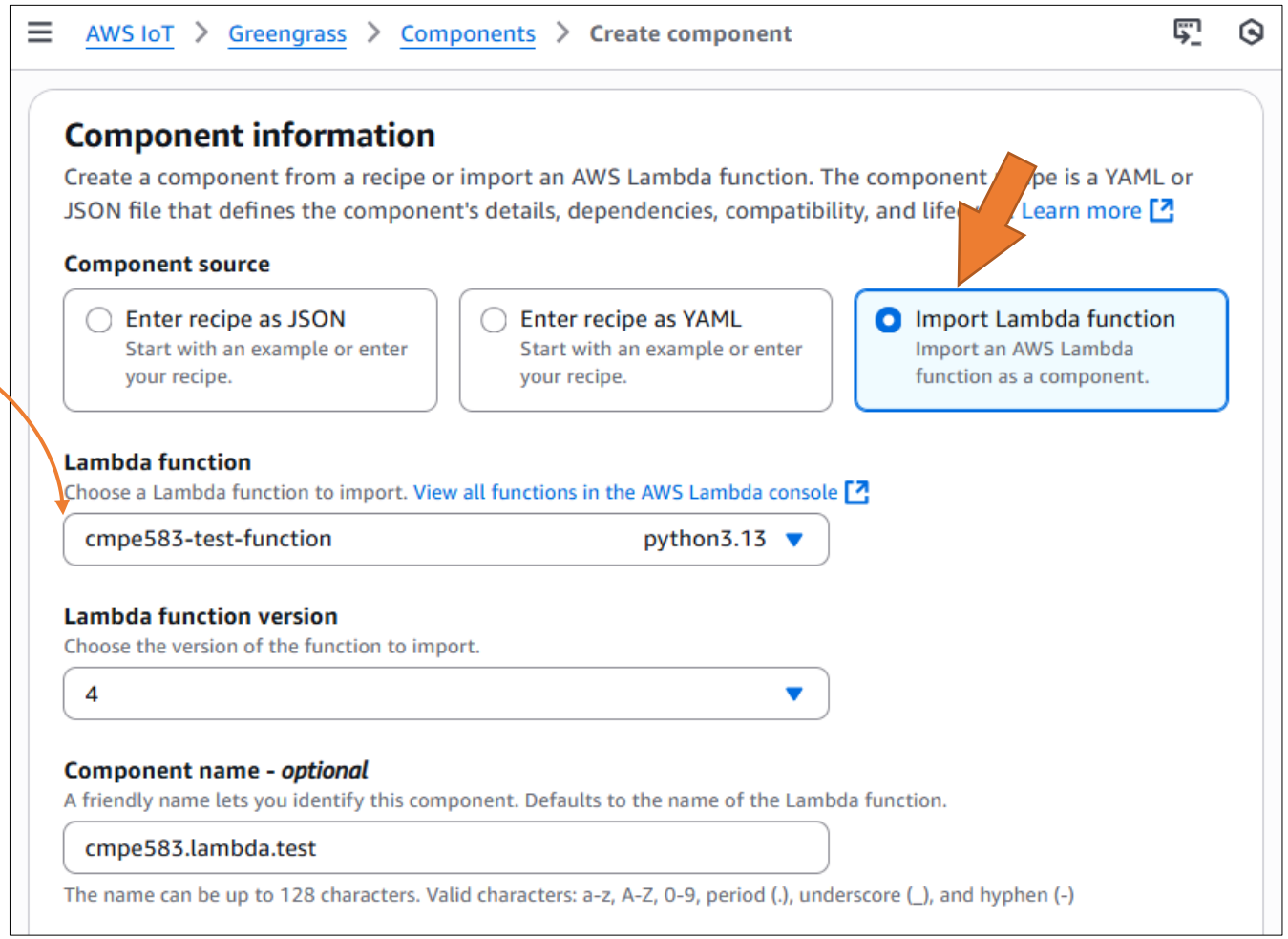
- Our lambda function is very simple for testing purpose, but if you have a more complex function that needs to access AWS resources (server, S3 file, database, etc.), you will need to configure the lambda's role accordingly.



Step 2: Create a Custom Component

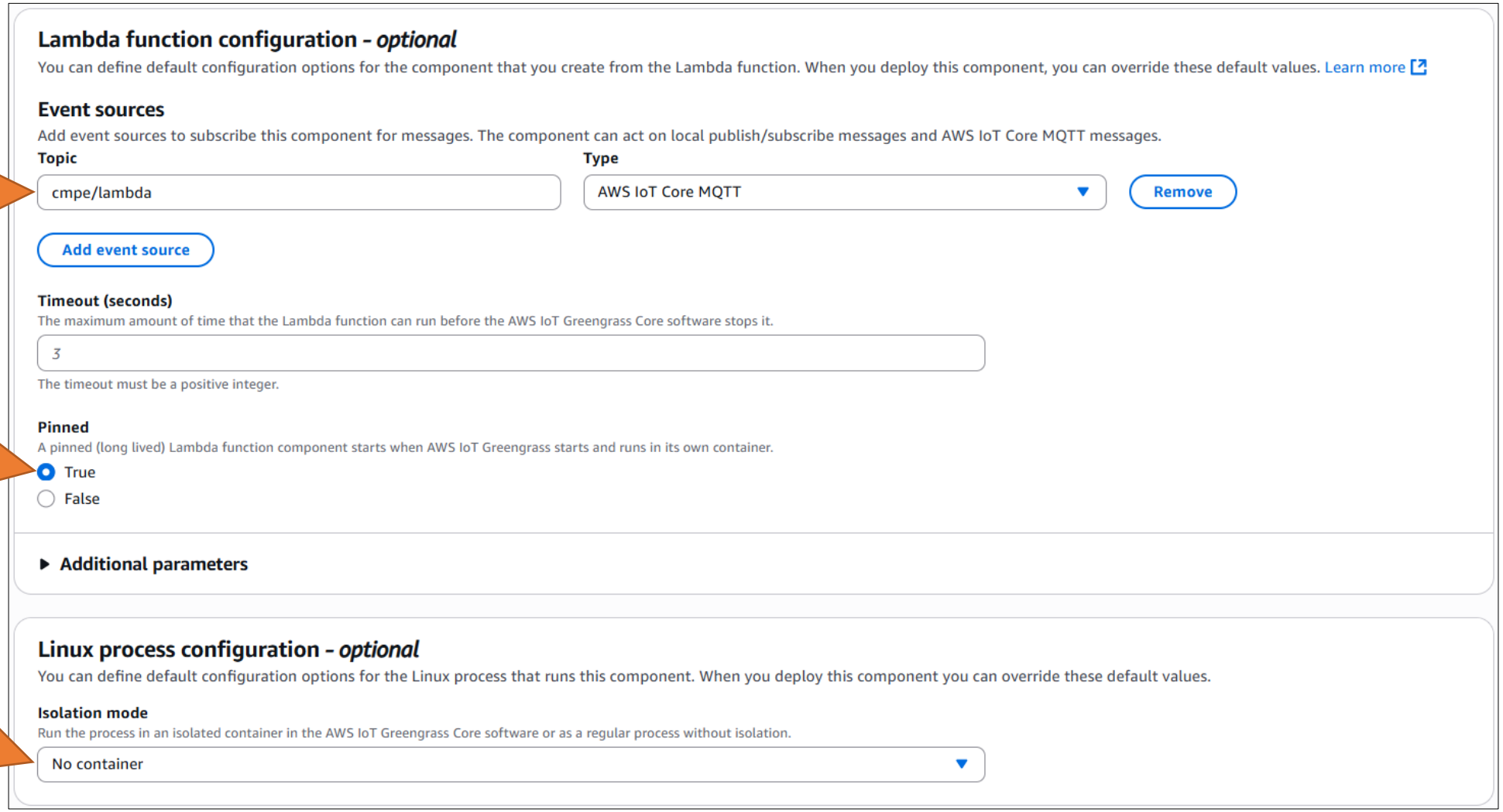
Import Lambda Function





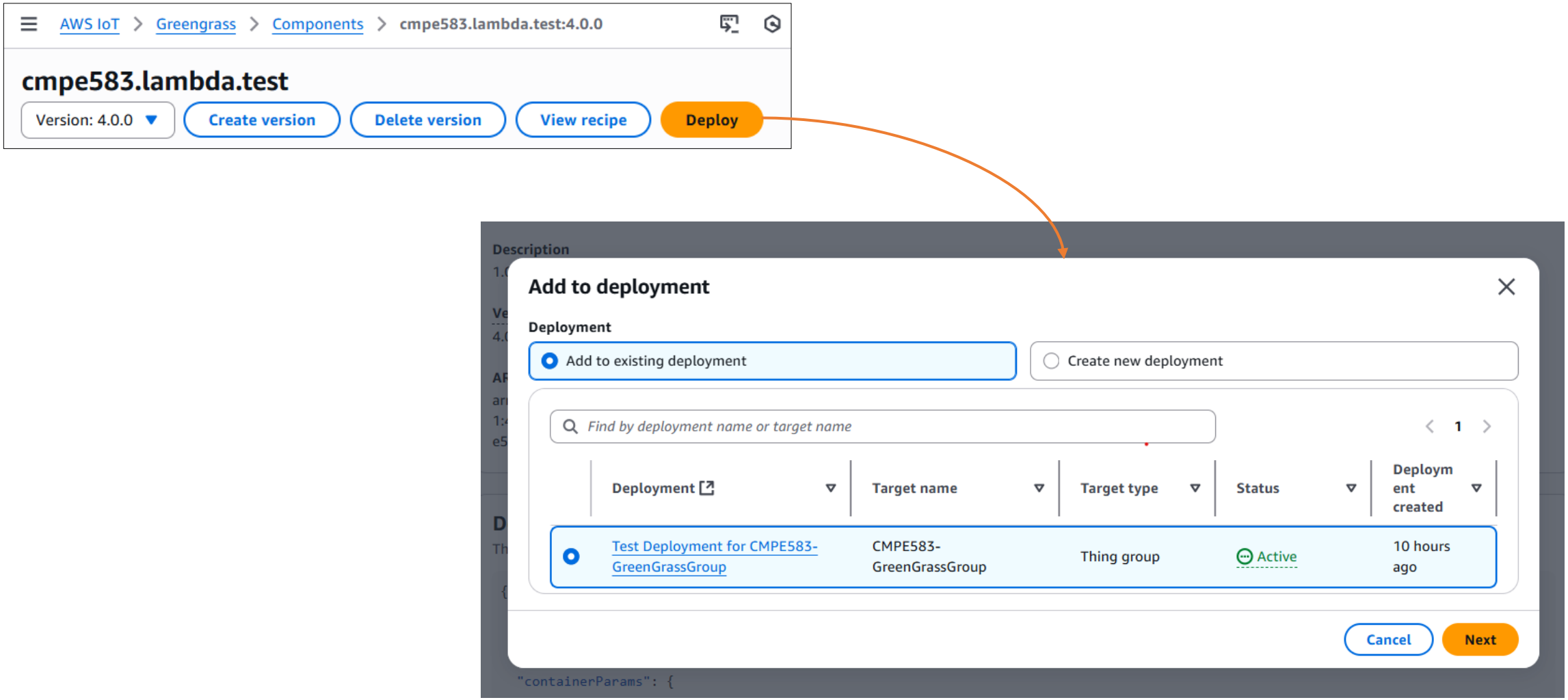
Step 2: Create a Custom Component

Configure Event Source (to Trigger Function) and Lifecycle



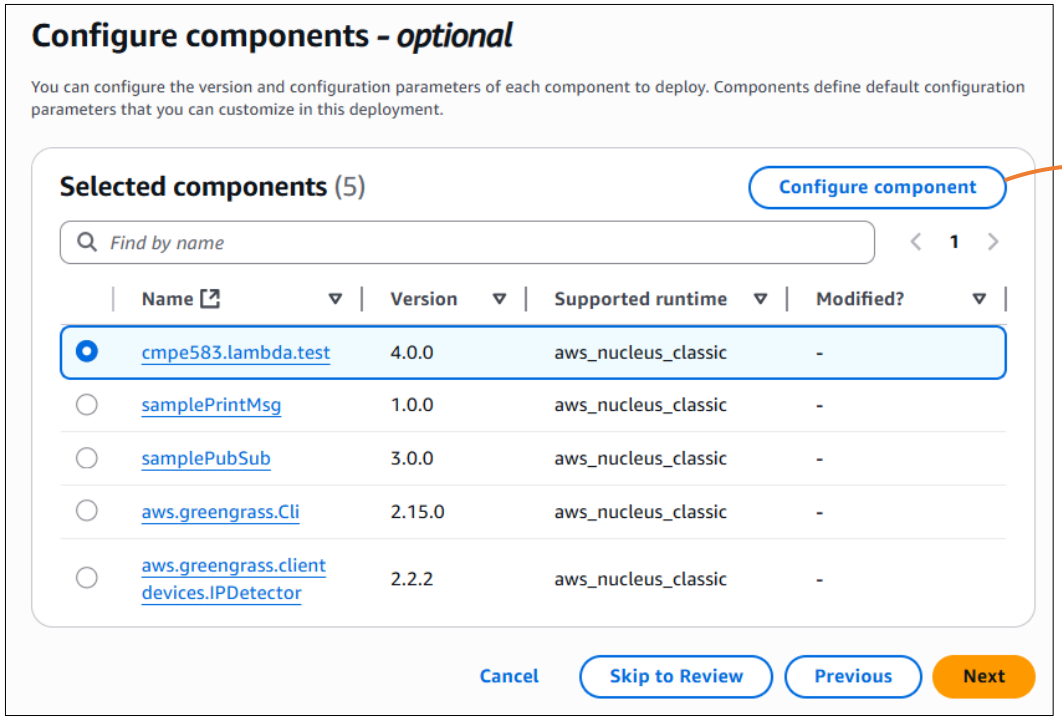
Step 3: Deploy Your Component

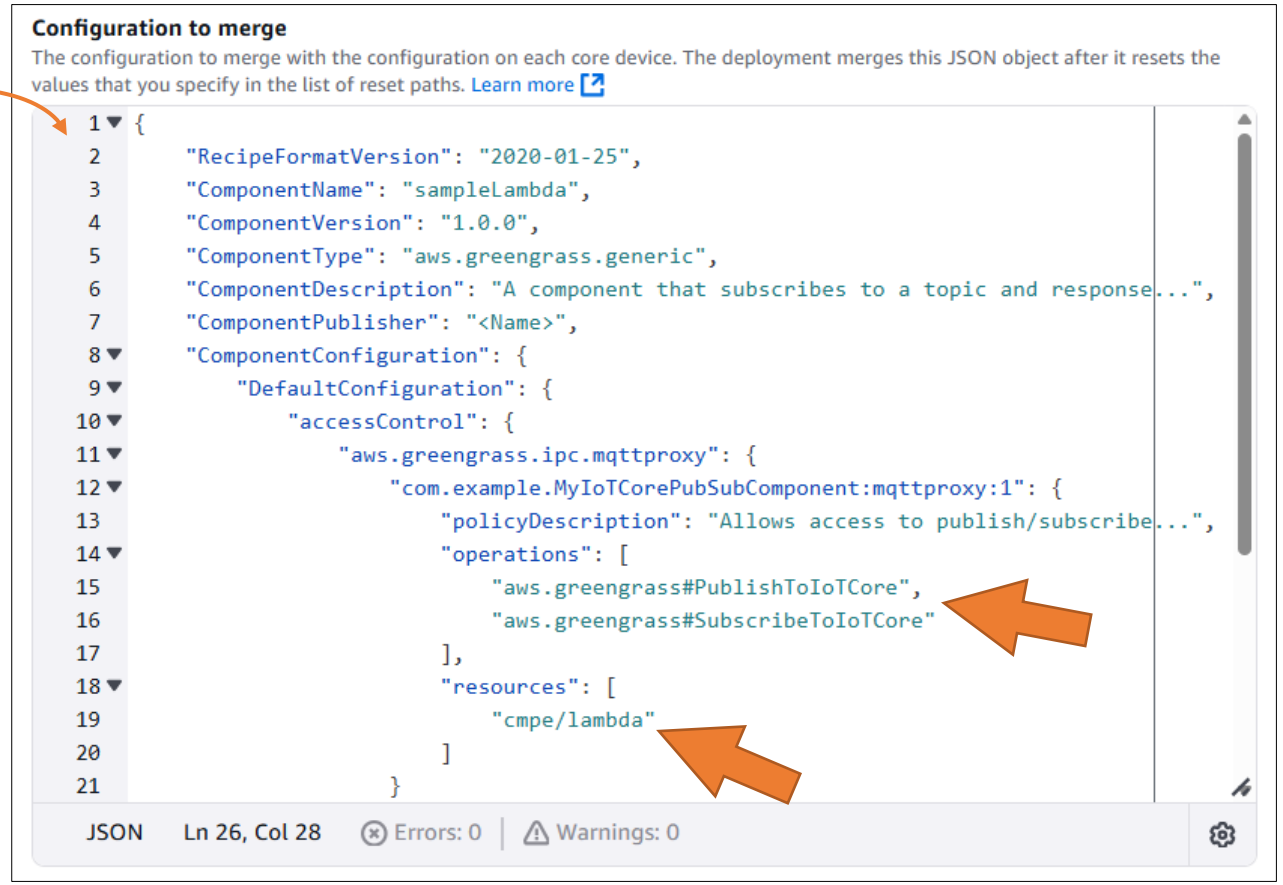
Add Component to Existing Deployment



Step 3: Deploy Your Component

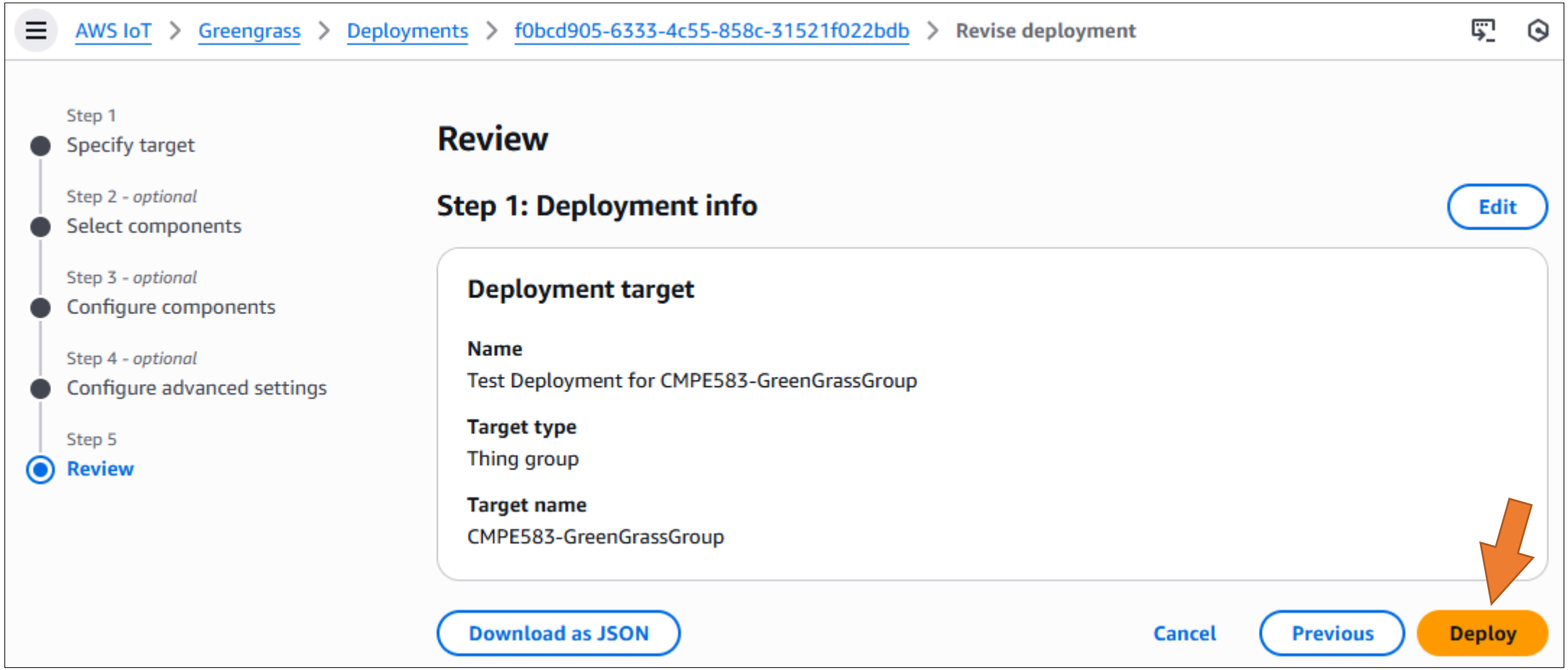
Configure Component to Merge Recipe





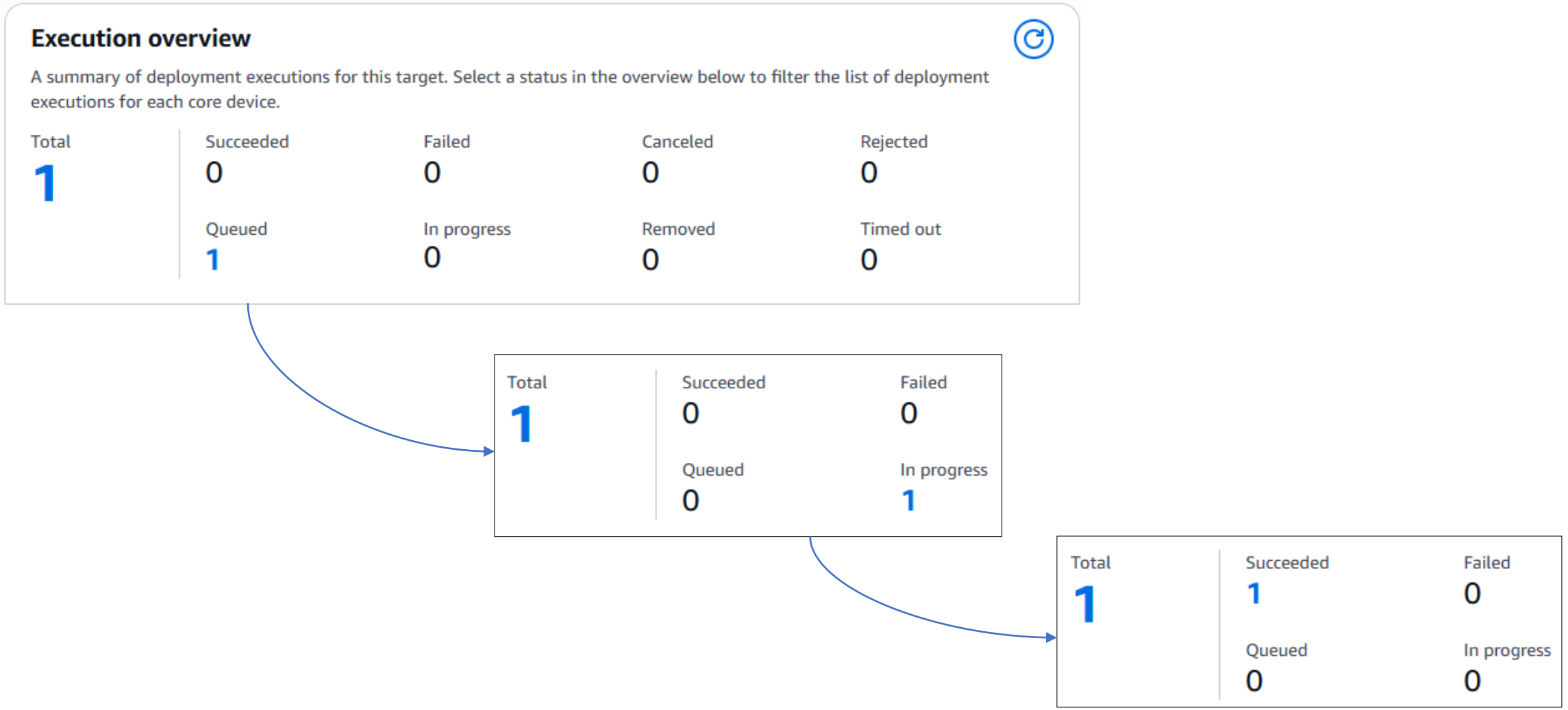
Step 3: Deploy the Component

Review & Deploy



Step 4: Verify the Component

Check AWS IoT Greengrass Console



Test Your Deployment

Via AWS MQTT Test Client

MQTT test client Info

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

Connection details Connected — Disconnect

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

Subscribe to a topic | Publish to a topic

Topic name

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

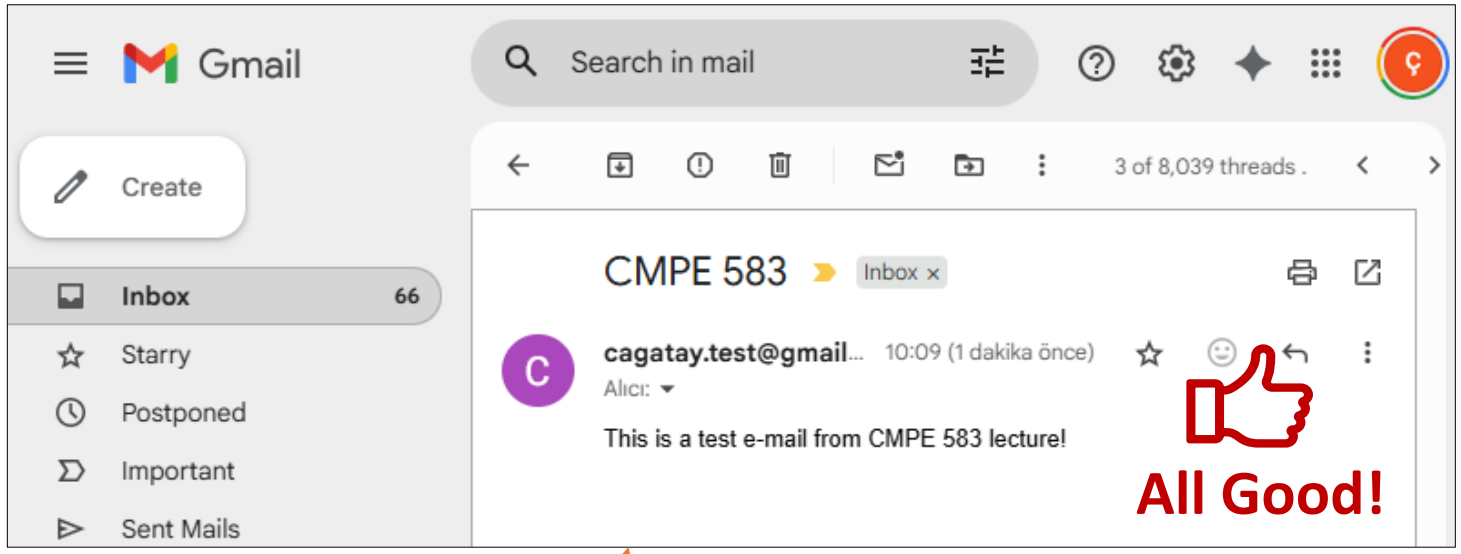
cmpe/lambda

Message payload

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

Additional configuration

Publish



Test Your Deployment

Verify From The Core Device's Logs

```
root@kubuntu:/home/cagatay/Projects/greengrass-demo# tail -n 100 -f /greengrass/v2/logs/cmpe583.lamda.test.log
2023-10-29T14:51:24.654Z [INFO] (pool-2-thread-52) cmpe583.lamda.test: shell-runner-start. {scriptName=services.cmpe583.lam
da.test.lifecycle.startup.script, serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=STARTING, command=["/gree
ngrass/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.la
mda.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, service
Name=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. {servic
eInstance=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-cen
tral-1:465822364162:function:cmpe583-test-function:3]. {serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=RUN
NING}
2023-10-29T14:51:25.052Z [INFO] (pool-2-thread-55) cmpe583.lamda.test: lambda_function.py:13,Sending test email.... {servic
eInstance=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!. {service
Instance=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.lam
da.test.lifecycle.shutdown.script, serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=STOPPING, command=["/gre
engrass/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 run
time.. {serviceInstance=1, serviceName=cmpe583.lamda.test, currentState=STOPPING}
```

Edge Computing with AWS IoT Greengrass - *Part #2*

Interact with Local IoT Devices

Local IoT Devices

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

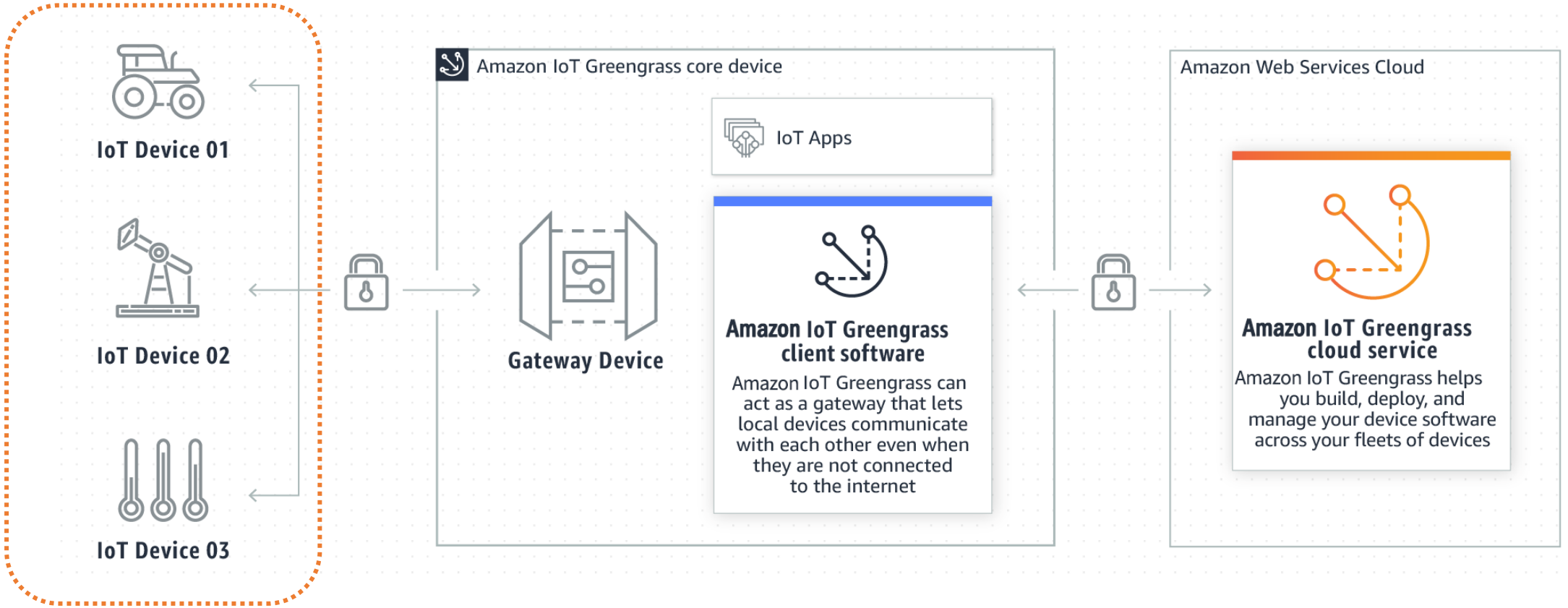


image source: https://aws.amazon.com/greengrass/

Connect Client Devices to Core Devices

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

AWS-Provided Client Device Components

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

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

Creating Client Devices

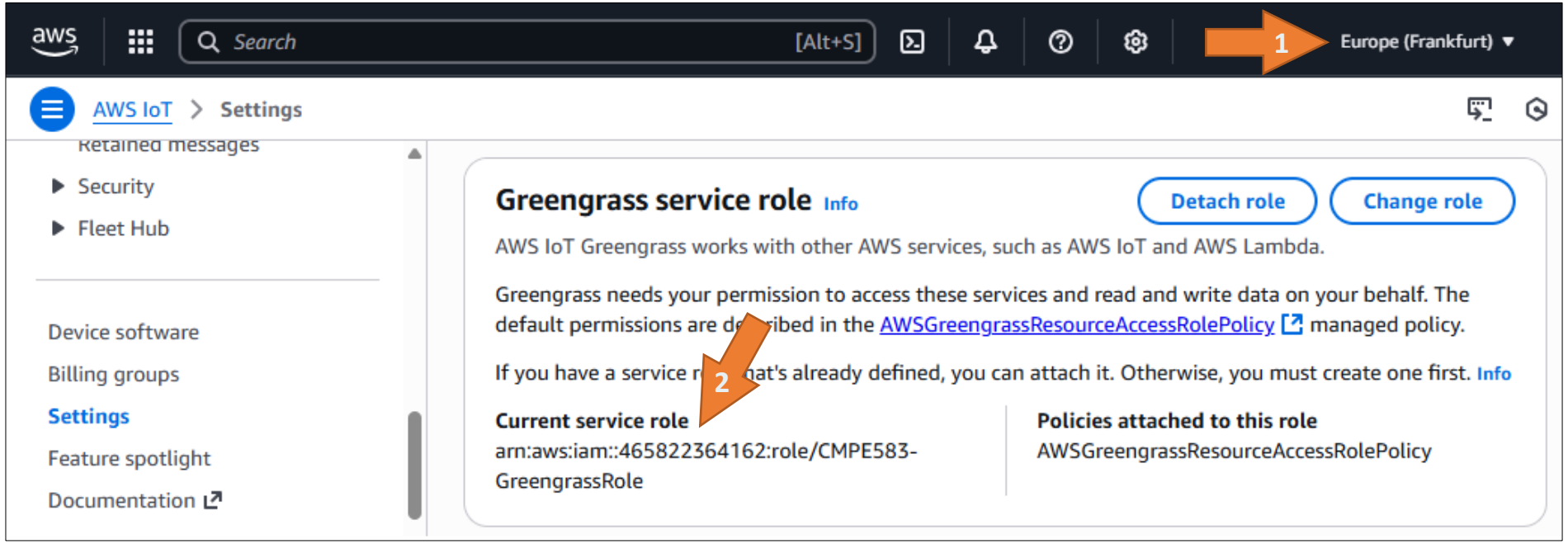
Steps for Creating a Client Device

1. Configure the Greengrass Service Role
2. Configure the AWS IoT Thing Policy
3. Create the Client Device (Thing) via AWS IoT Console
4. Download Certificates and Keys (needed for secure auth. and communication)

Step 1: Configure the Greengrass Service Role

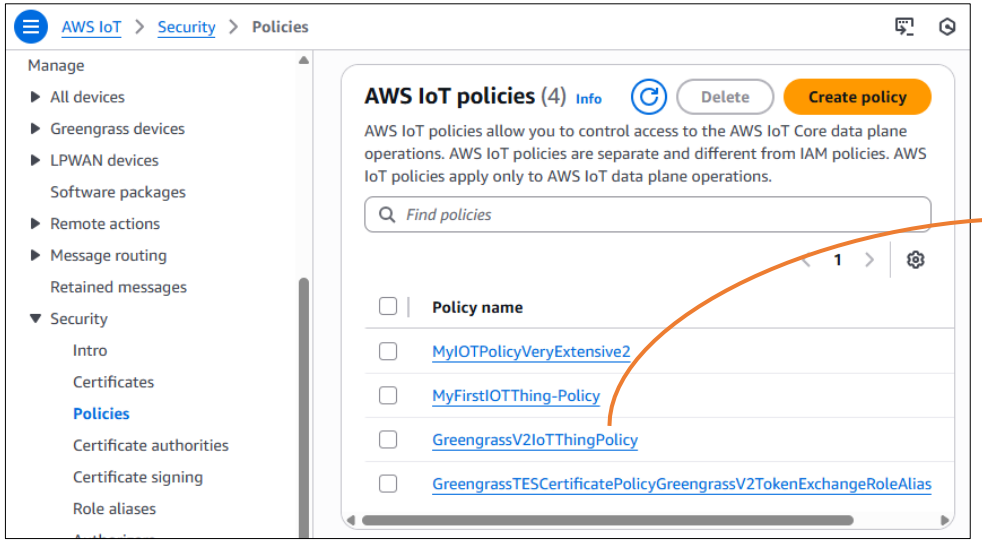
Verify If Greengrass Service Role is Attached

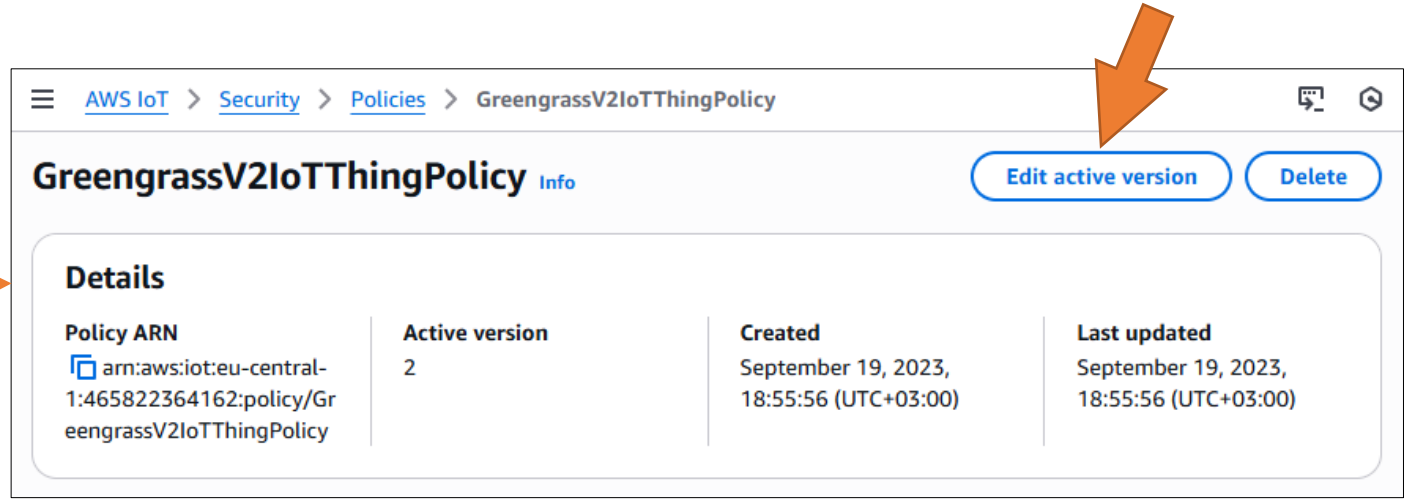
- The Greengrass service role is an IAM service role that authorizes AWS IoT Greengrass to access resources from AWS services on your behalf.
- Check whether the Greengrass service role is set up in AWS IoT console's settings.



Step 2: Configure the AWS IoT Thing Policy

Review the Policy



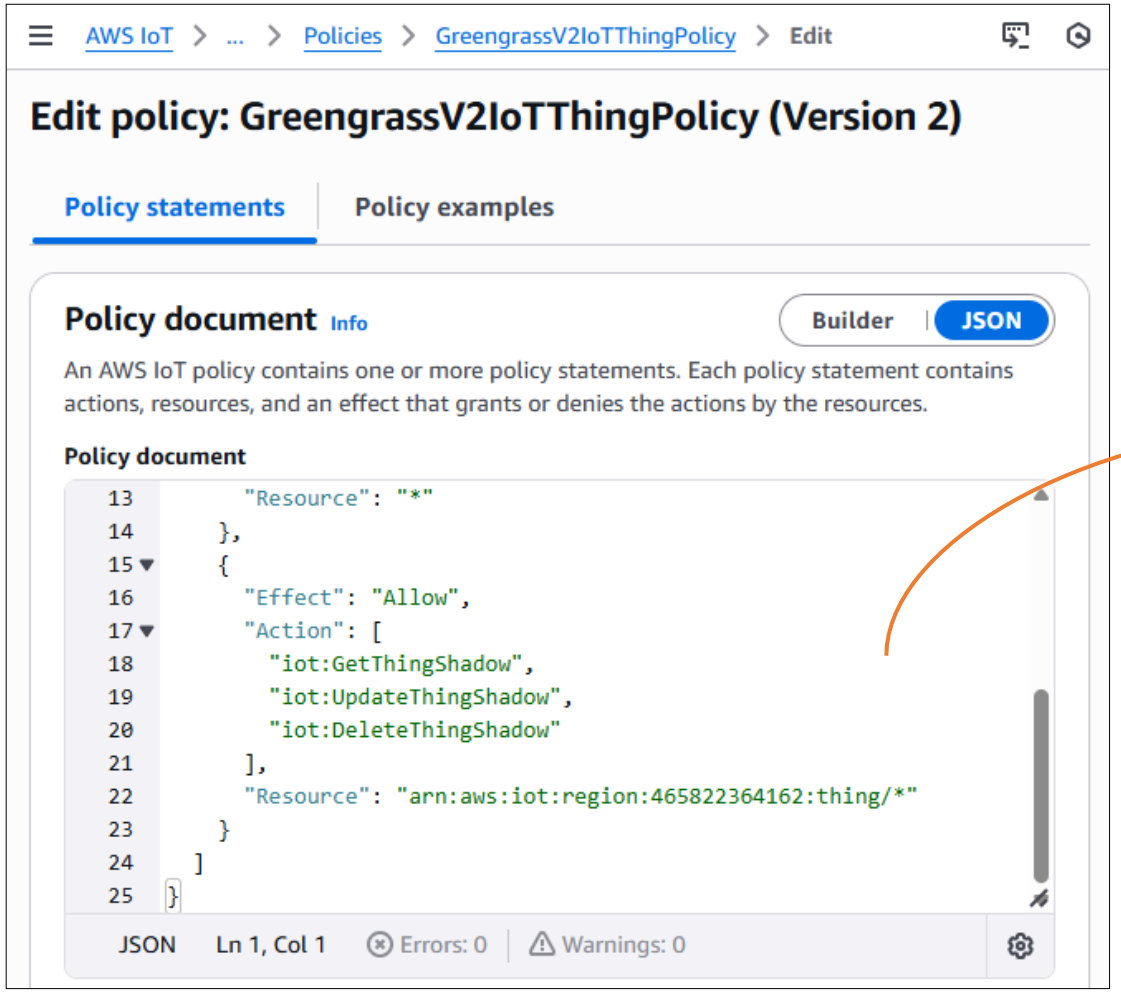


➢ Review the policy for required permissions, and add any required permissions that are missing:

- greengrass:PutCertificateAuthorities
- greengrass:VerifyClientDeviceIdentity
- greengrass:VerifyClientDeviceIoTCertificateAssociation
- greengrass:GetConnectivityInfo

Step 2: Configure the AWS IoT Thing Policy

Optional: Allow Core Device to Sync Shadows

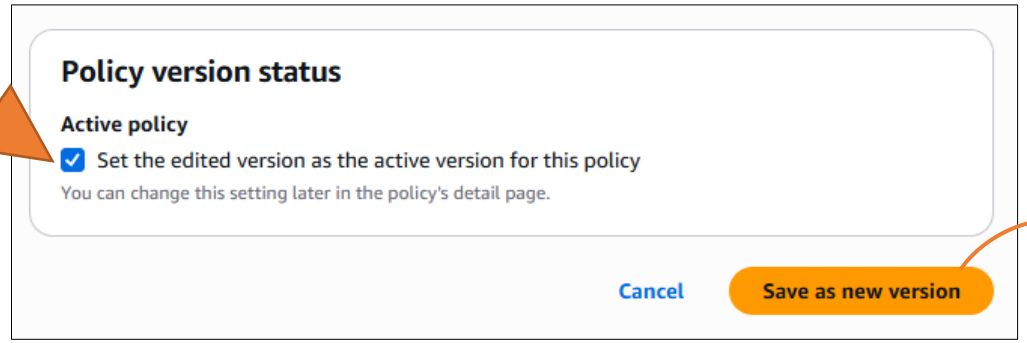


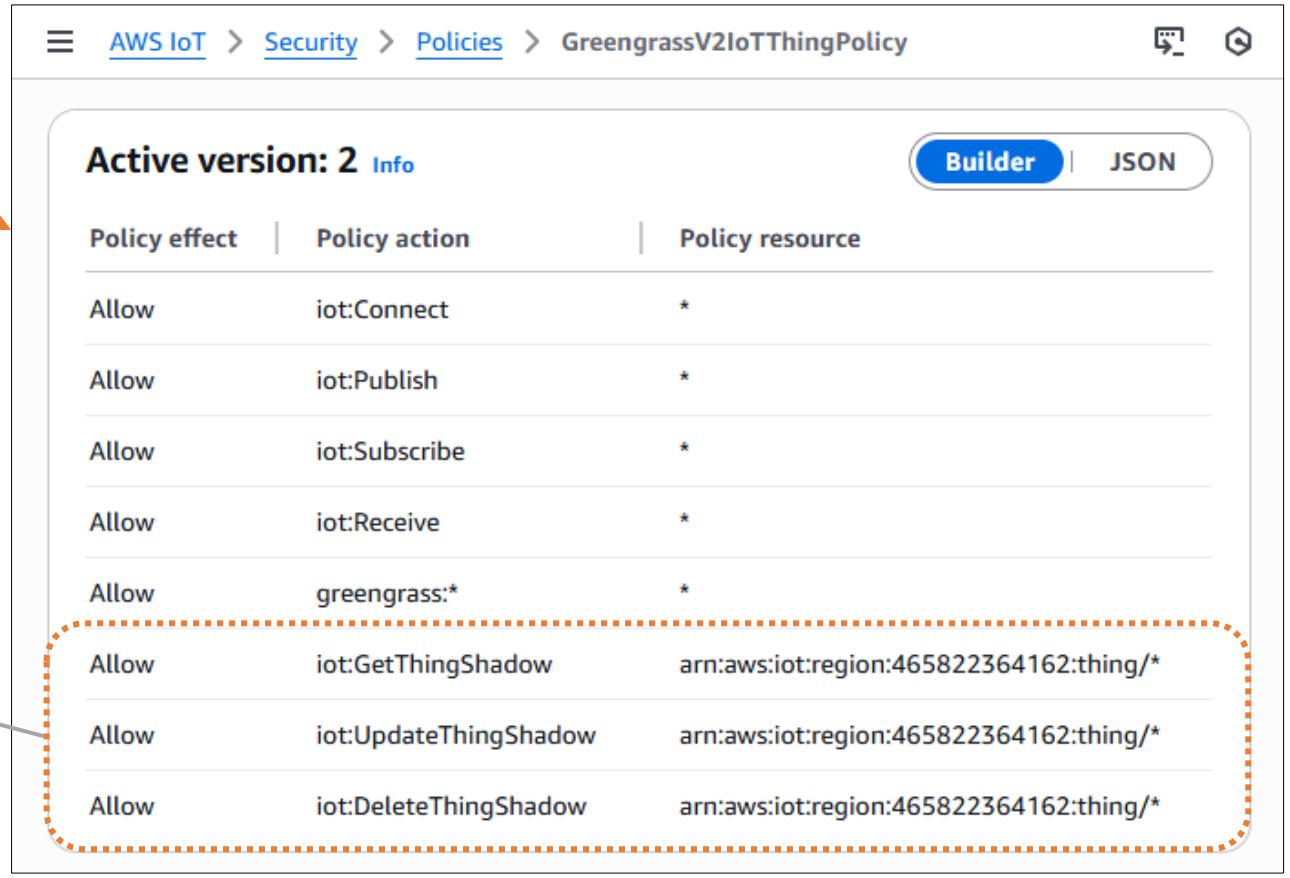
To allow the core device to sync shadows with AWS IoT Core, add the following statement to the policy:

```
{
  "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

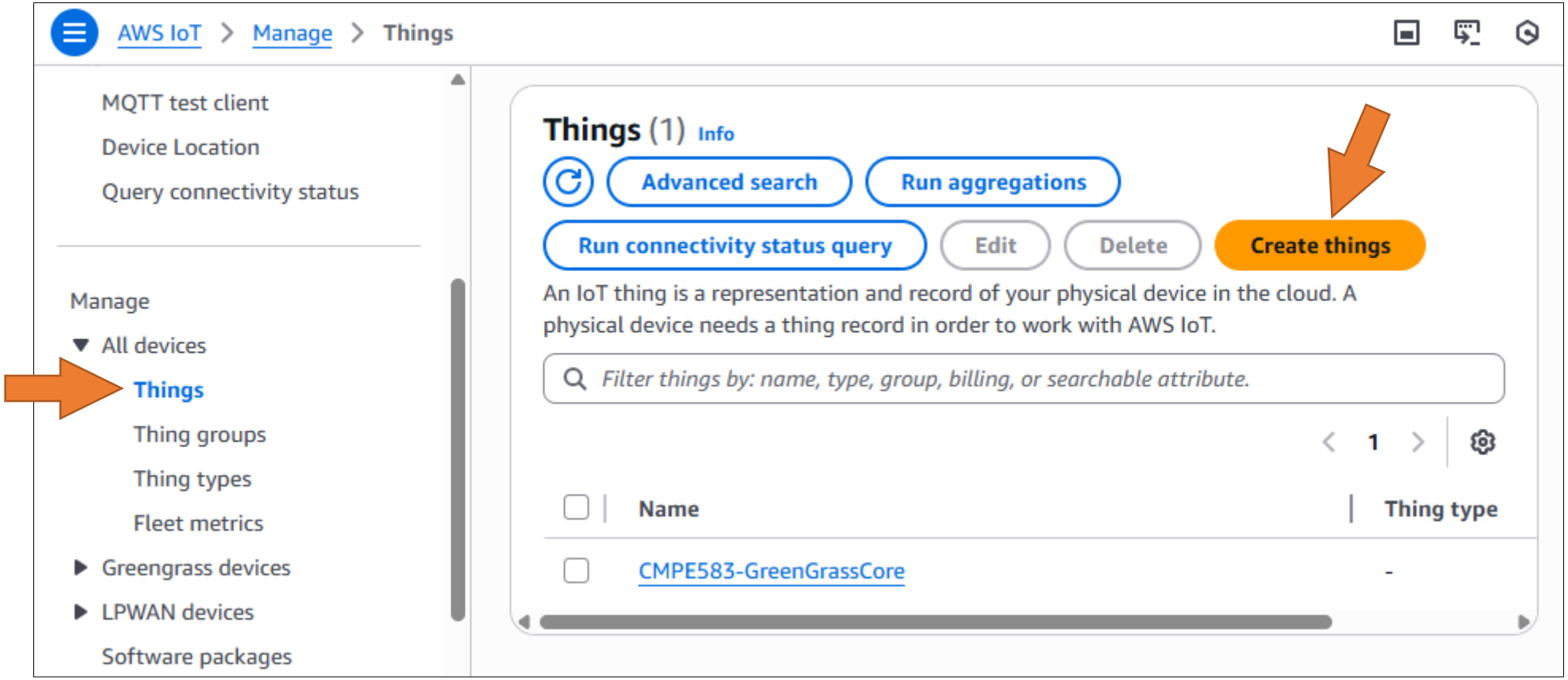




🚨 **Security Best Practice: Principle of Least Privilege**
Only add these permissions if your application specifically uses the Device Shadow service.

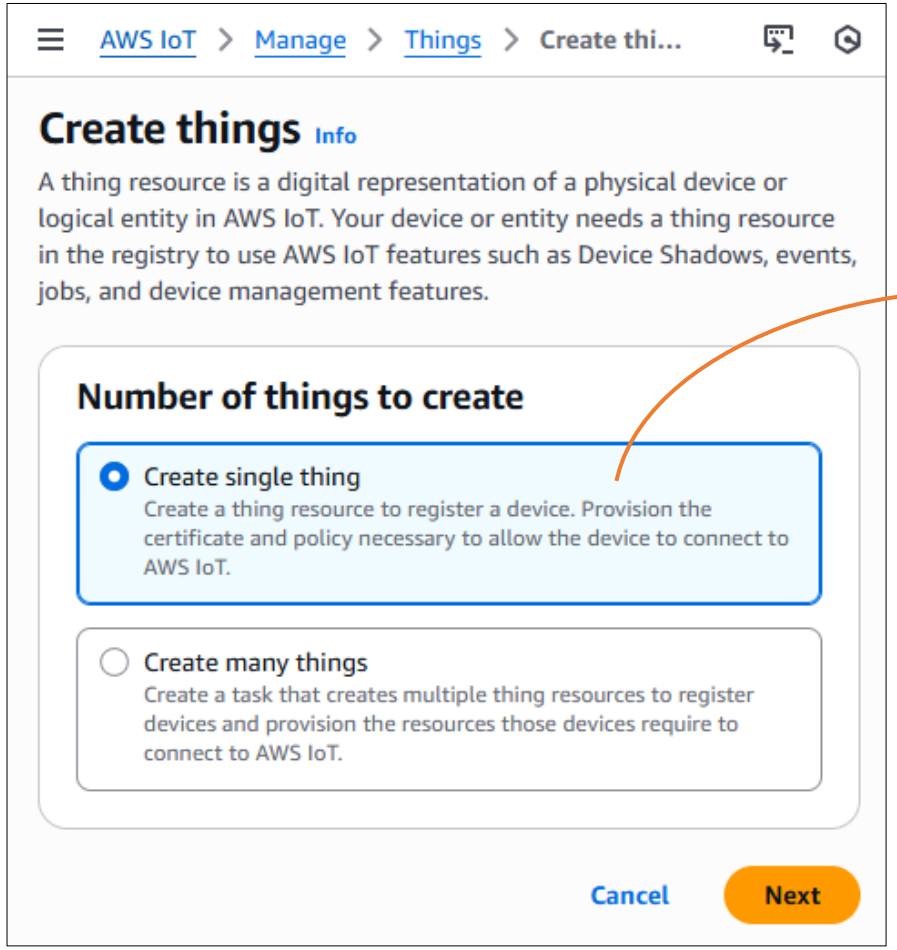
Step 3: Create Client Device (Thing)

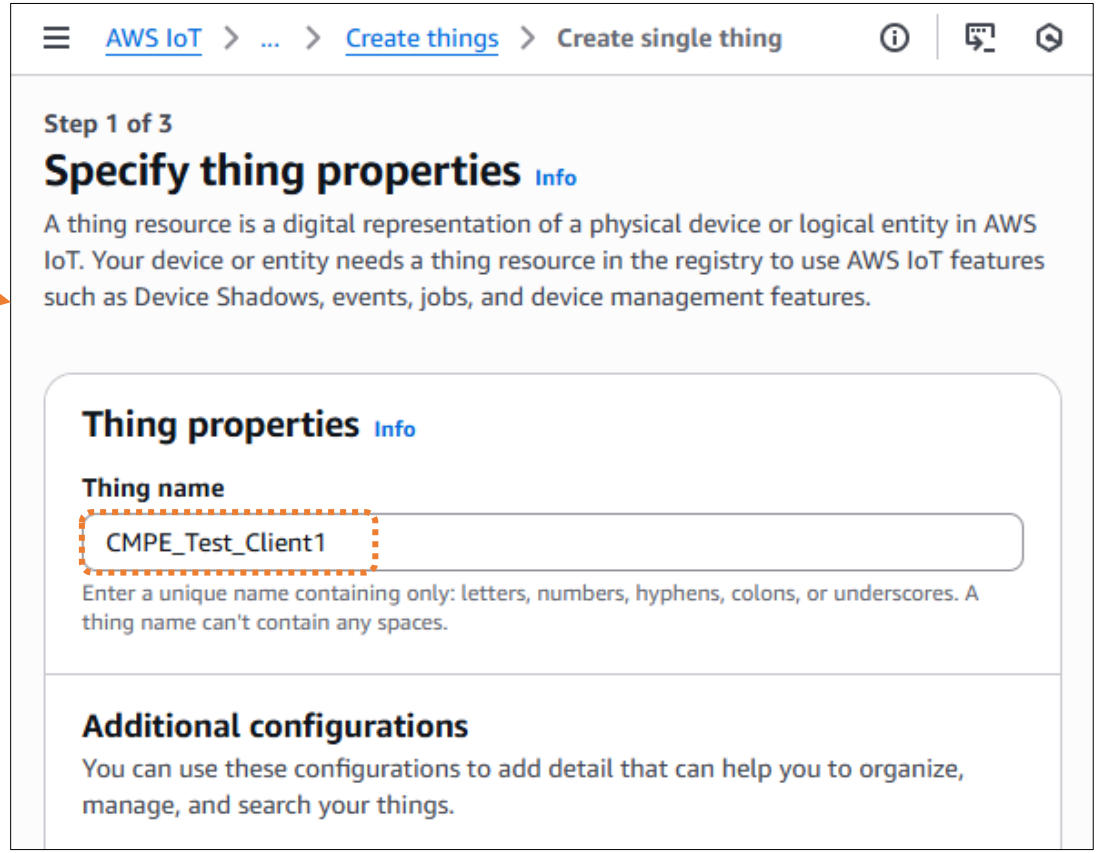
Via AWS IoT Console



Step 3: Create Client Device (Thing)

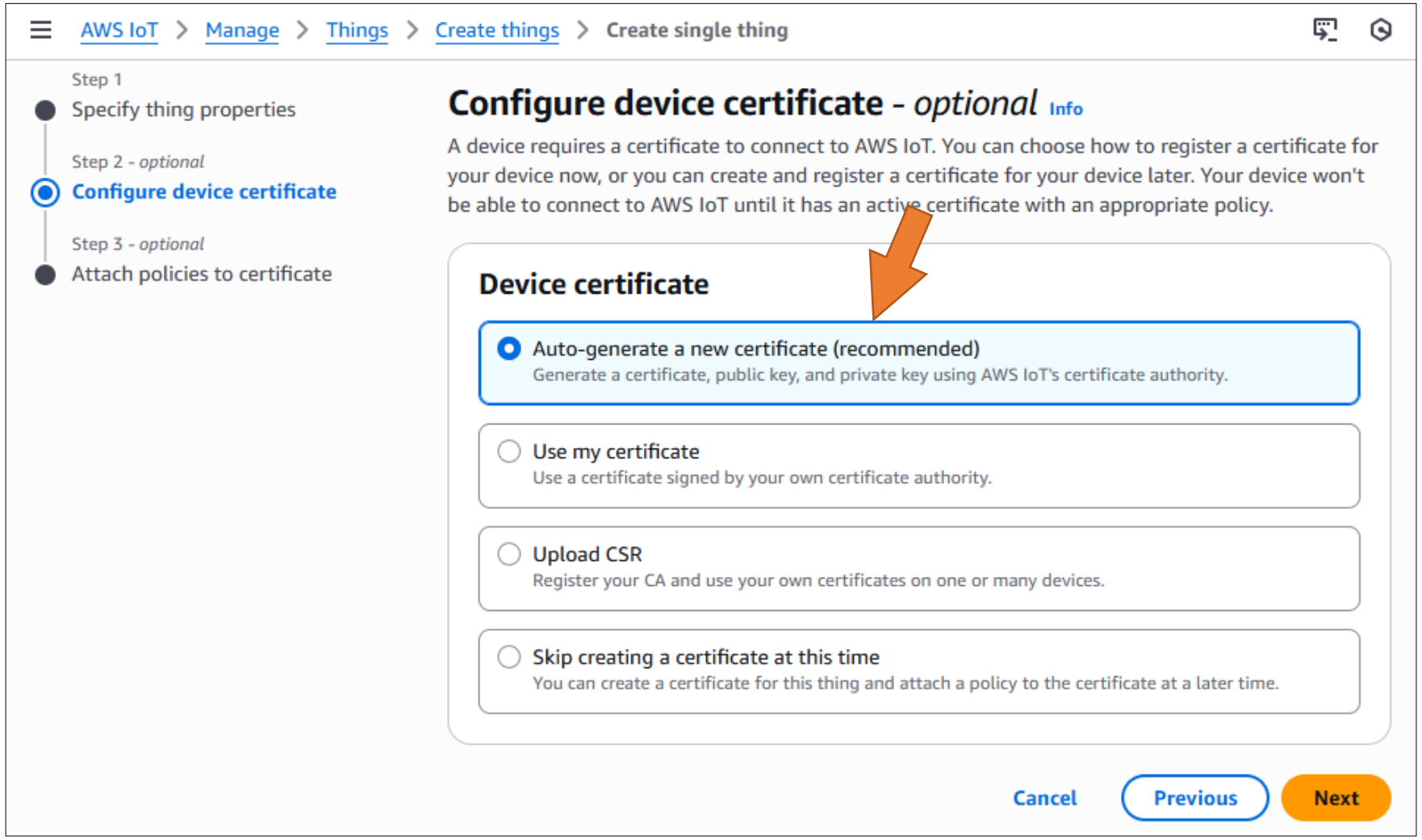
Via AWS IoT Console





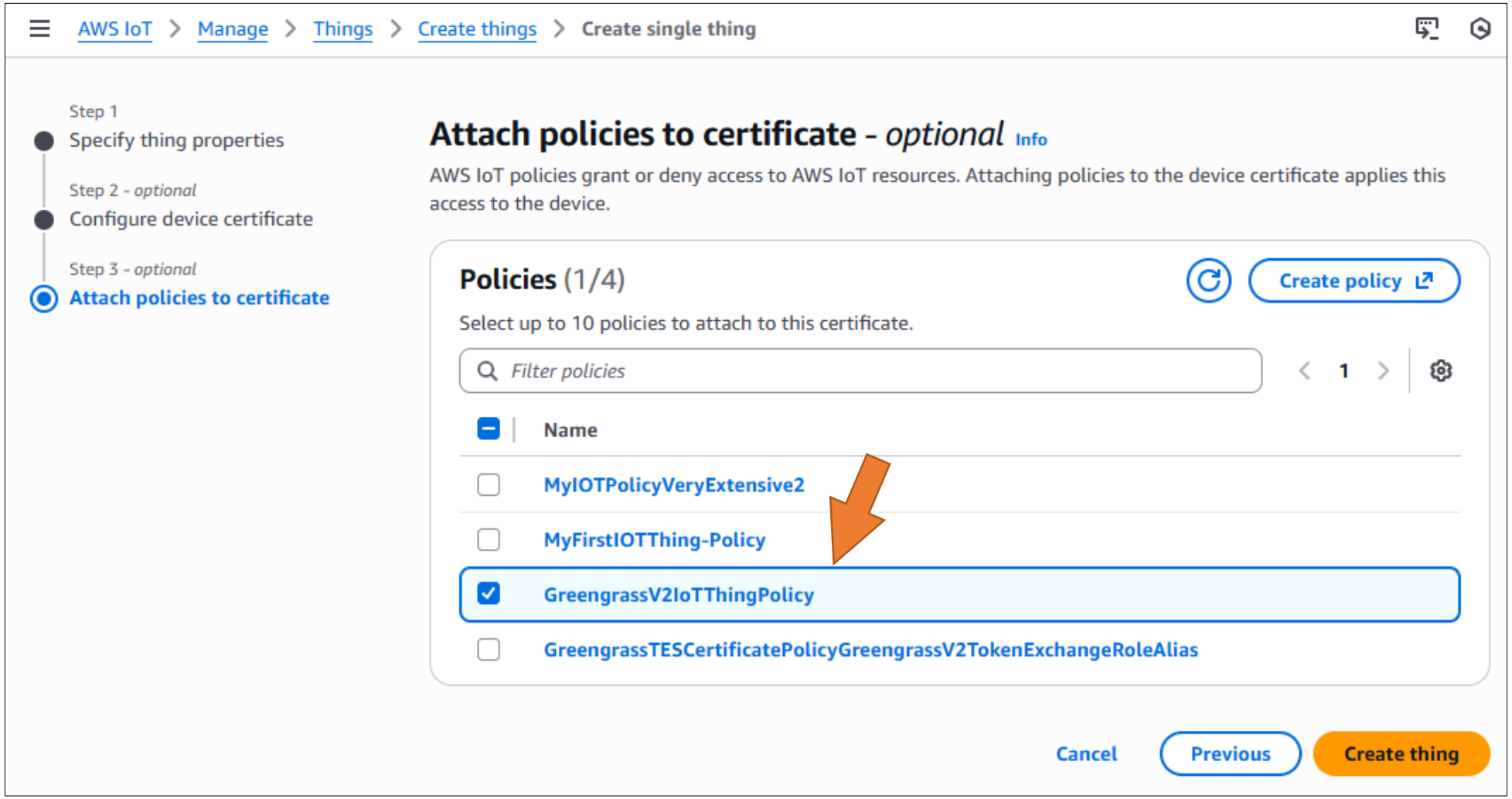
Step 3: Create Client Device (Thing)

Configure Device Certificate

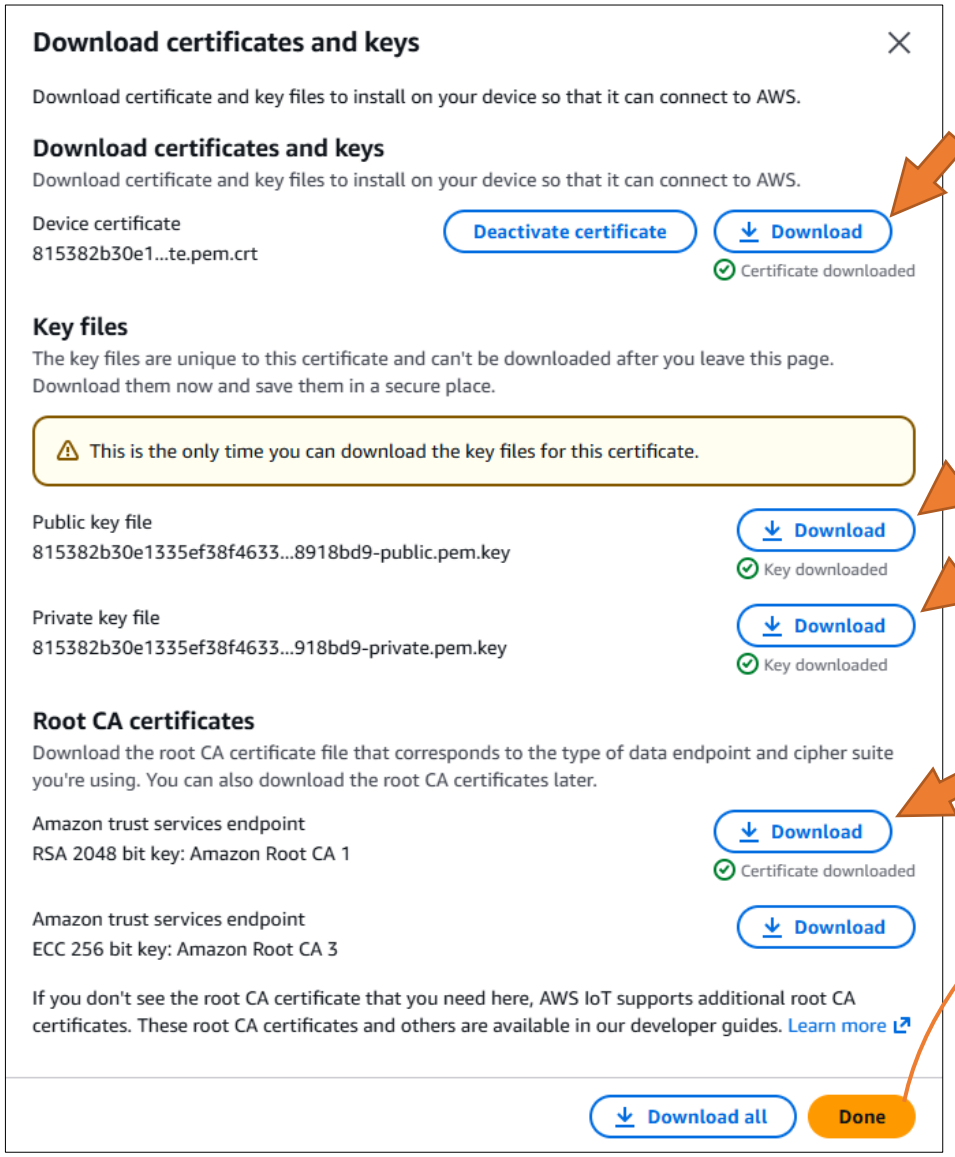


Step 3: Create Client Device (Thing)

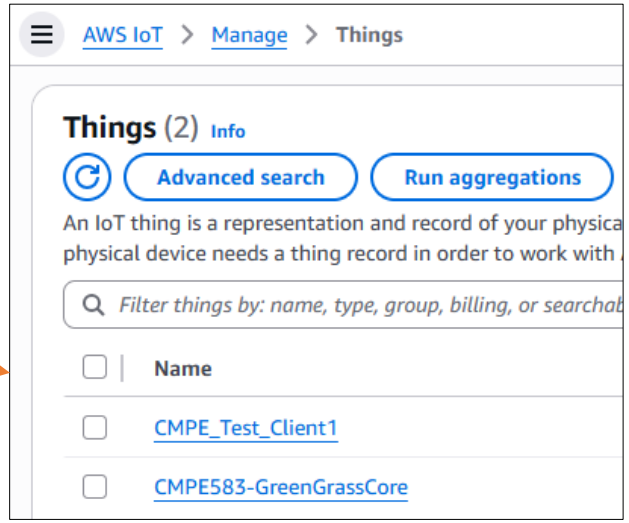
Configure Policy



Step 4: Download Certificates and Keys



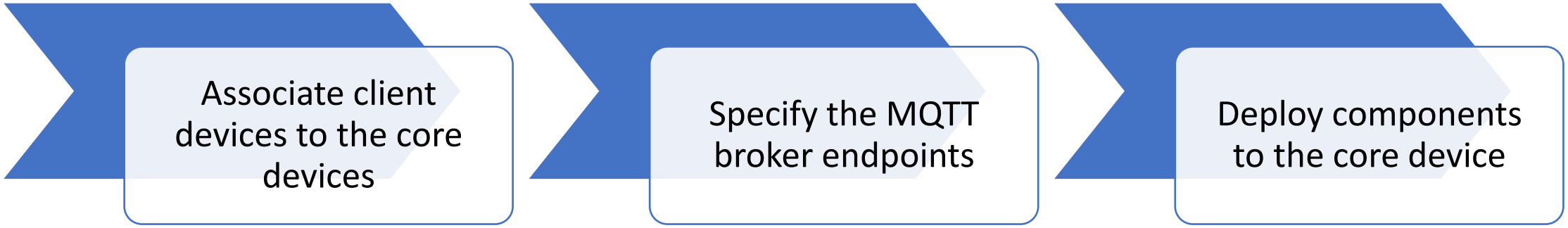
★ Key files must be downloaded in the last step of things creation!



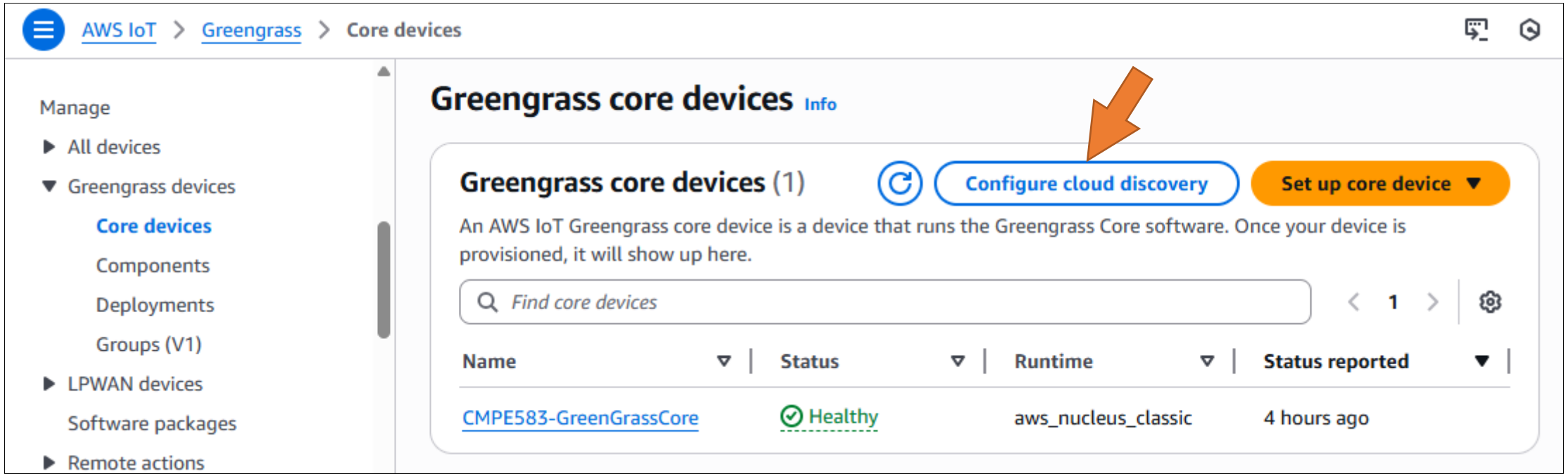
Configuring Cloud Discovery

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:

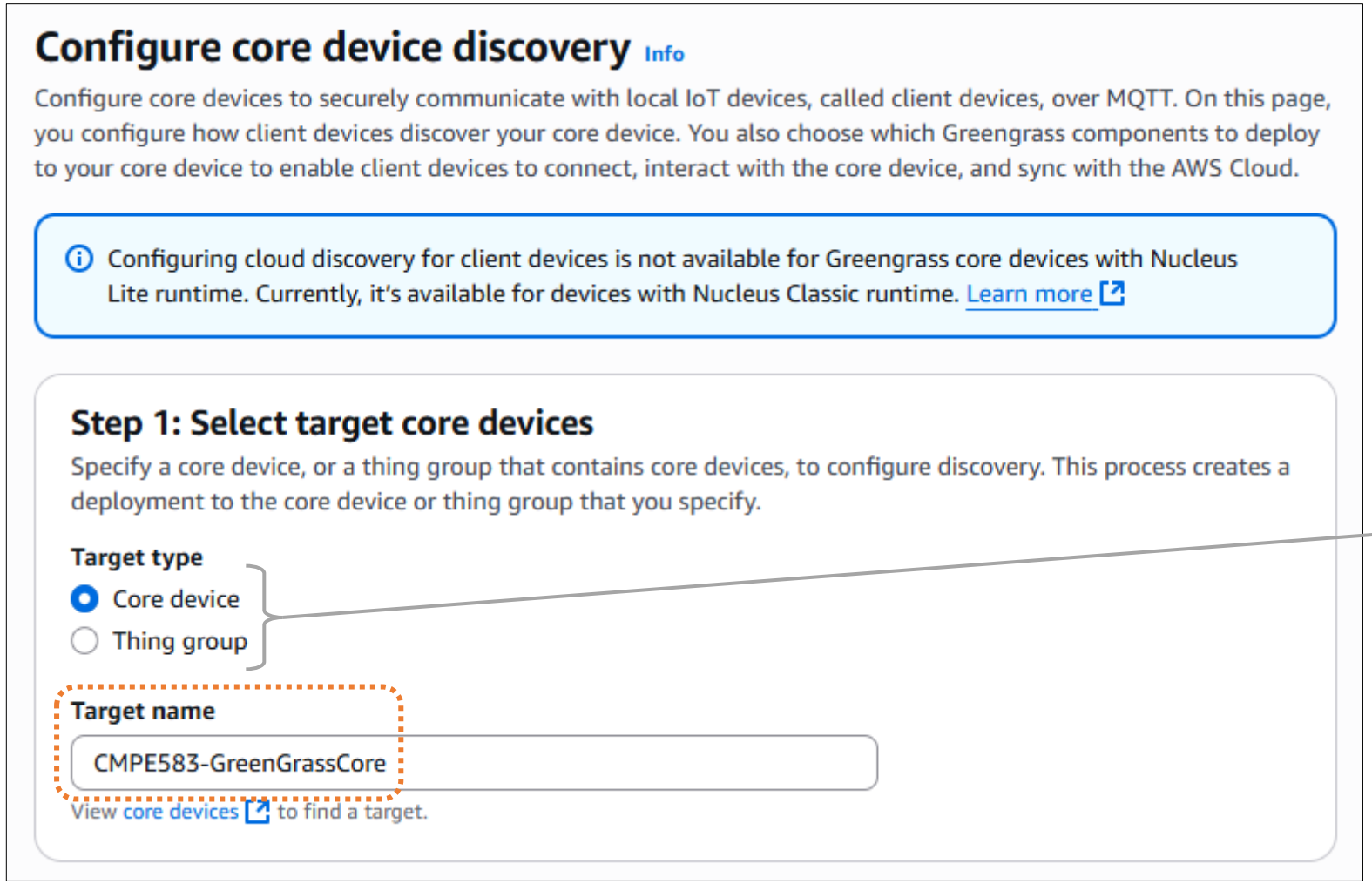


Step 1: Configure Cloud Discovery



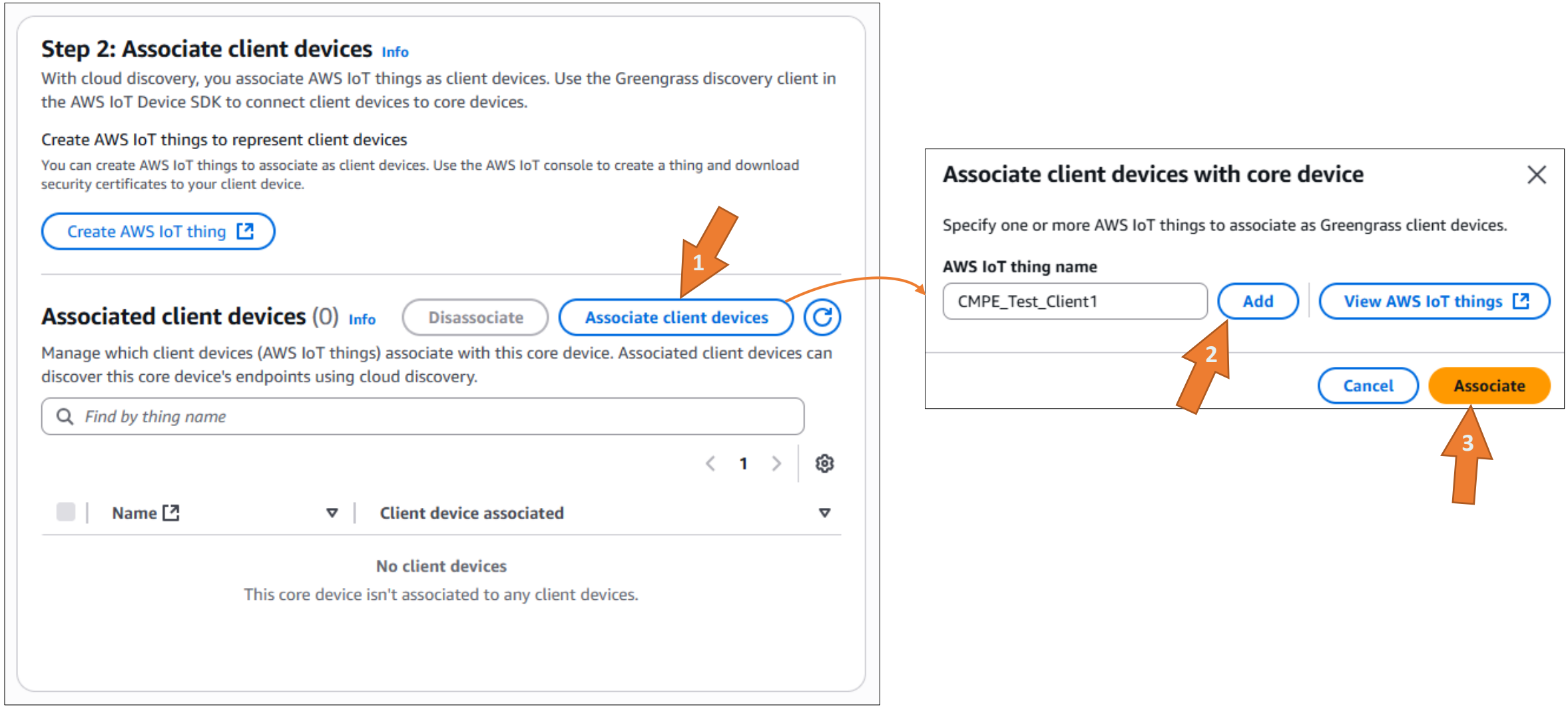
Step 1: Configure Cloud Discovery

Select Target Core Devices



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

Step 2: Associate Client Devices

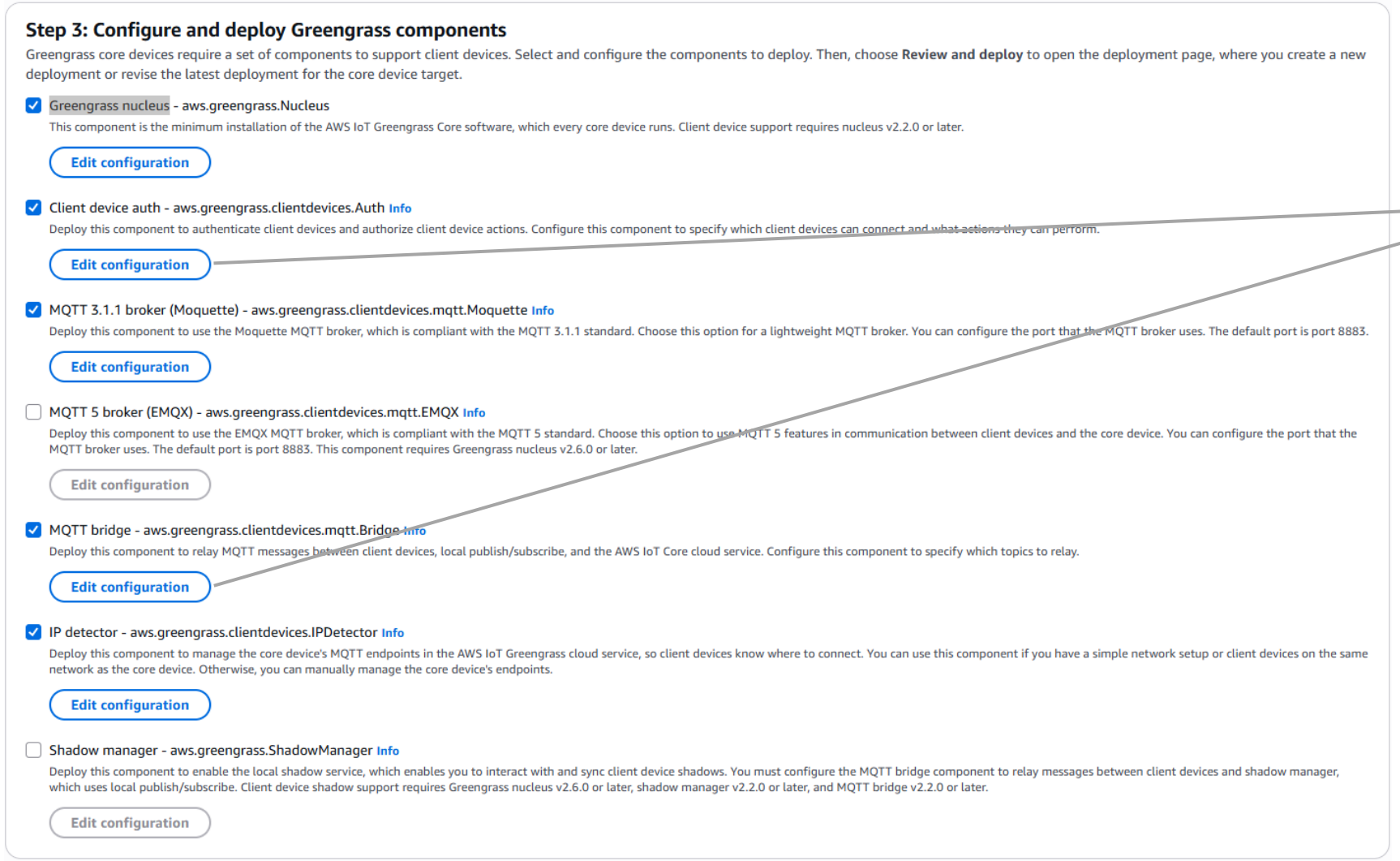


Step 3: Deploy Required Greengrass Components

- The following Greengrass components are required:
  - Greengrass nucleus (aws.greengrass.Nucleus)
  - Client device auth (aws.greengrass.clientdevices.Auth)
  - IP detector (aws.greengrass.clientdevices.IPDetector)
- You must also deploy ONE of the following local brokers:
  - MQTT 3.1.1 broker (Moquette) (aws.greengrass.clientdevices.mqtt.Moquette)
  - MQTT 5 broker (EMQX) (aws.greengrass.clientdevices.mqtt.EMQX)
- The following components are optional:
  - MQTT bridge (aws.greengrass.clientdevices.mqtt.Bridge)
  - Shadow manager (aws.greengrass.ShadowManager)

Step 3: Deploy Required Greengrass Components

Configure Required Greengrass Components

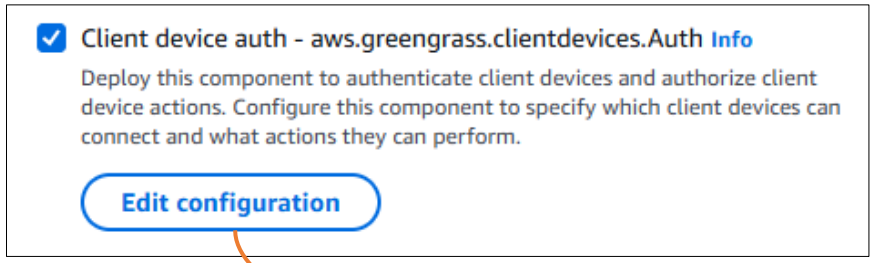


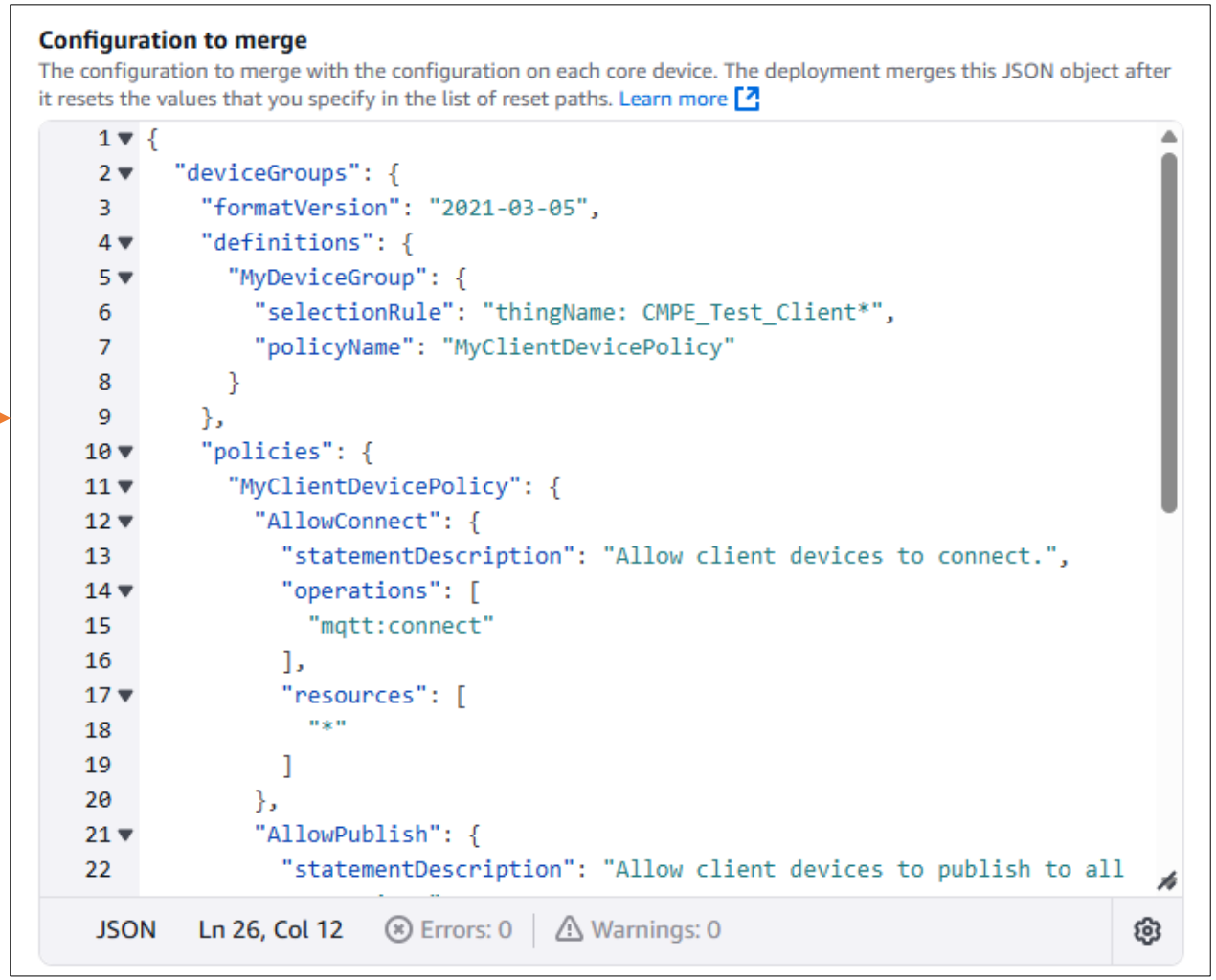
MQTT bridge and client device auth components should be configured before the deployment!

★ 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

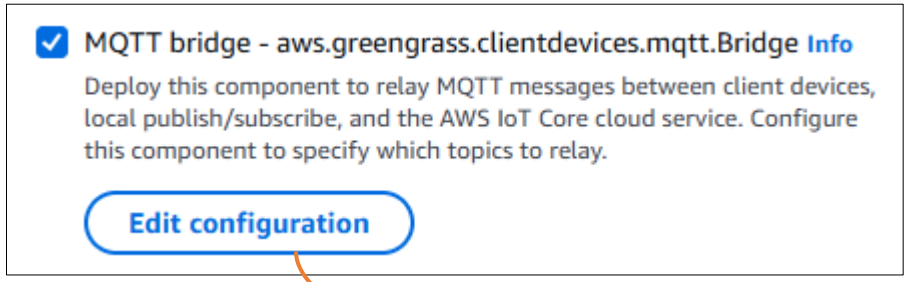


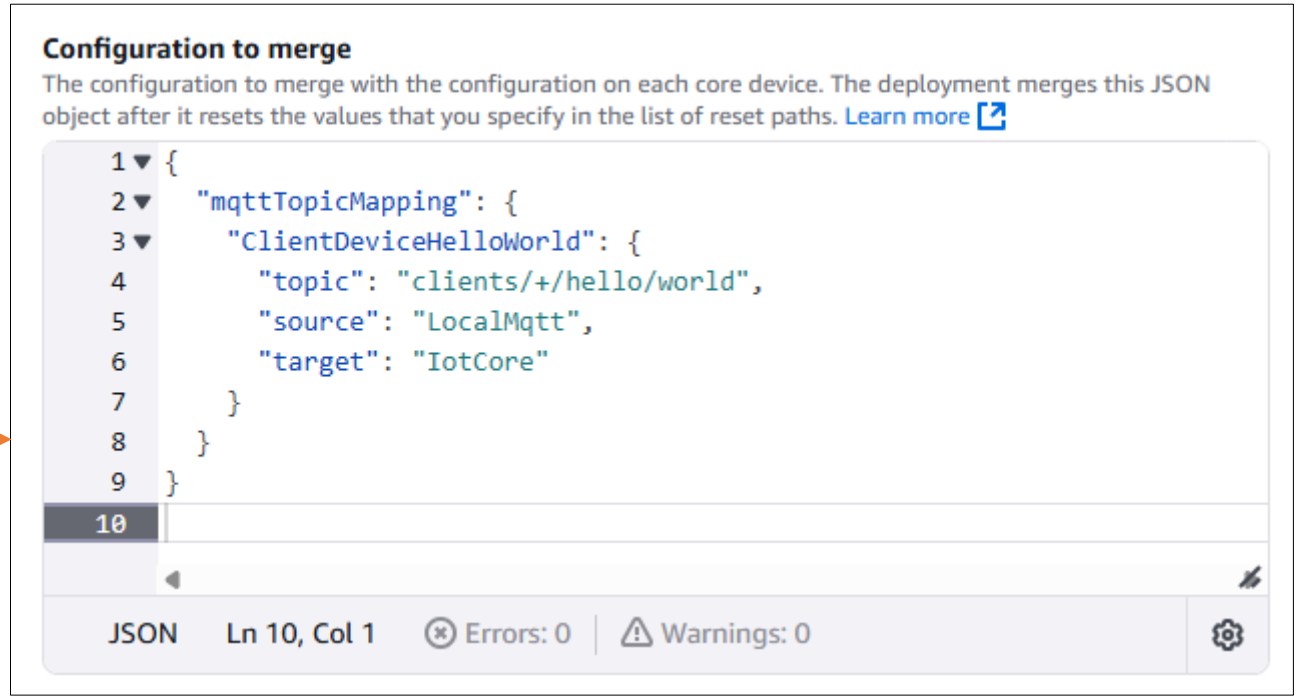


In this example, permissive policy is used to allow all operations. To use a more restrictive policy, check out the policy file in github: ***cmpe583-ClientDeviceAuth.Json***

Step 3: Deploy Required Greengrass Components

Customize aws.greengrass.clientdevices.mqtt.Bridge

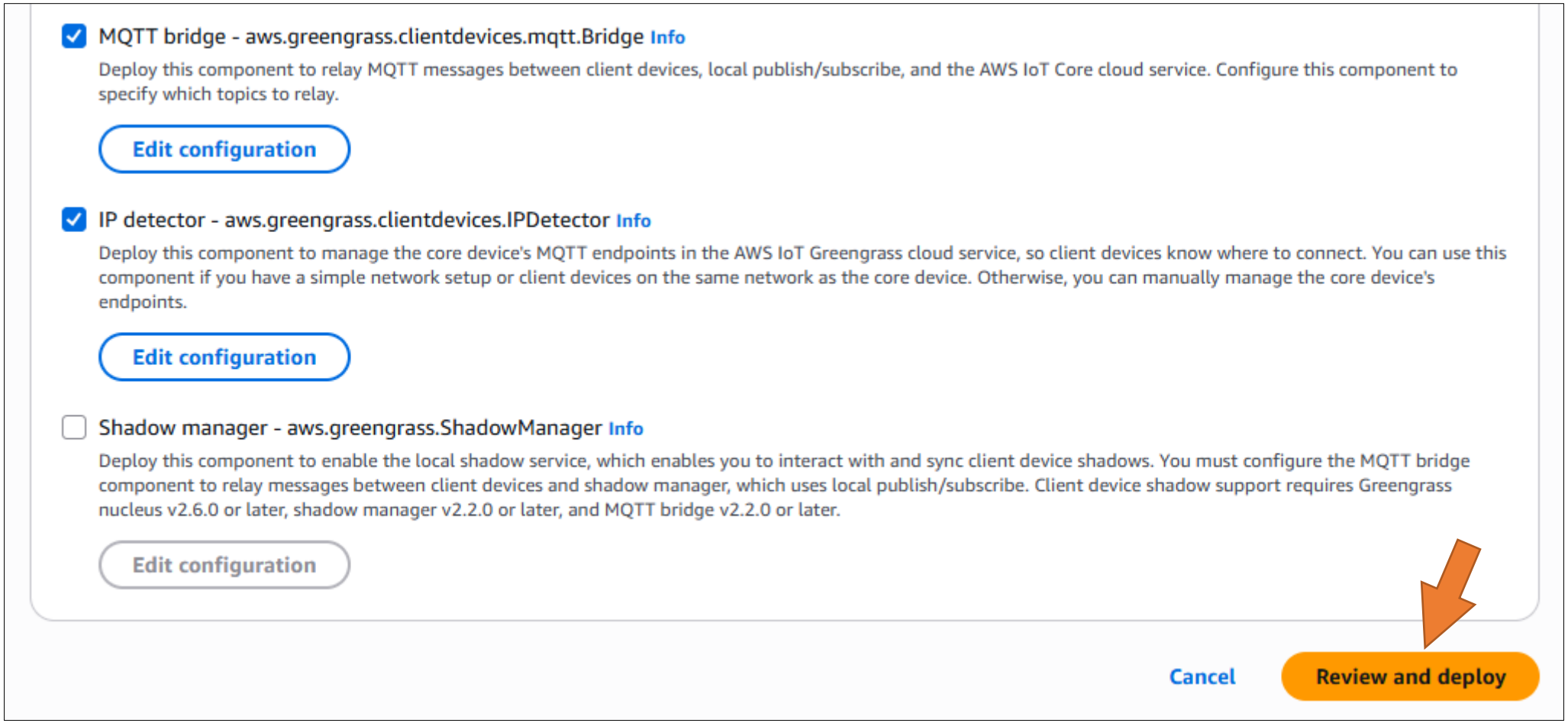




- Relay MQTT messages on the clients/+/hello/world topic filter **from client** devices **to the AWS IoT Core** cloud service.
- For example, this topic filter matches the clients/CMPE_Test_Client1/hello/world topic.

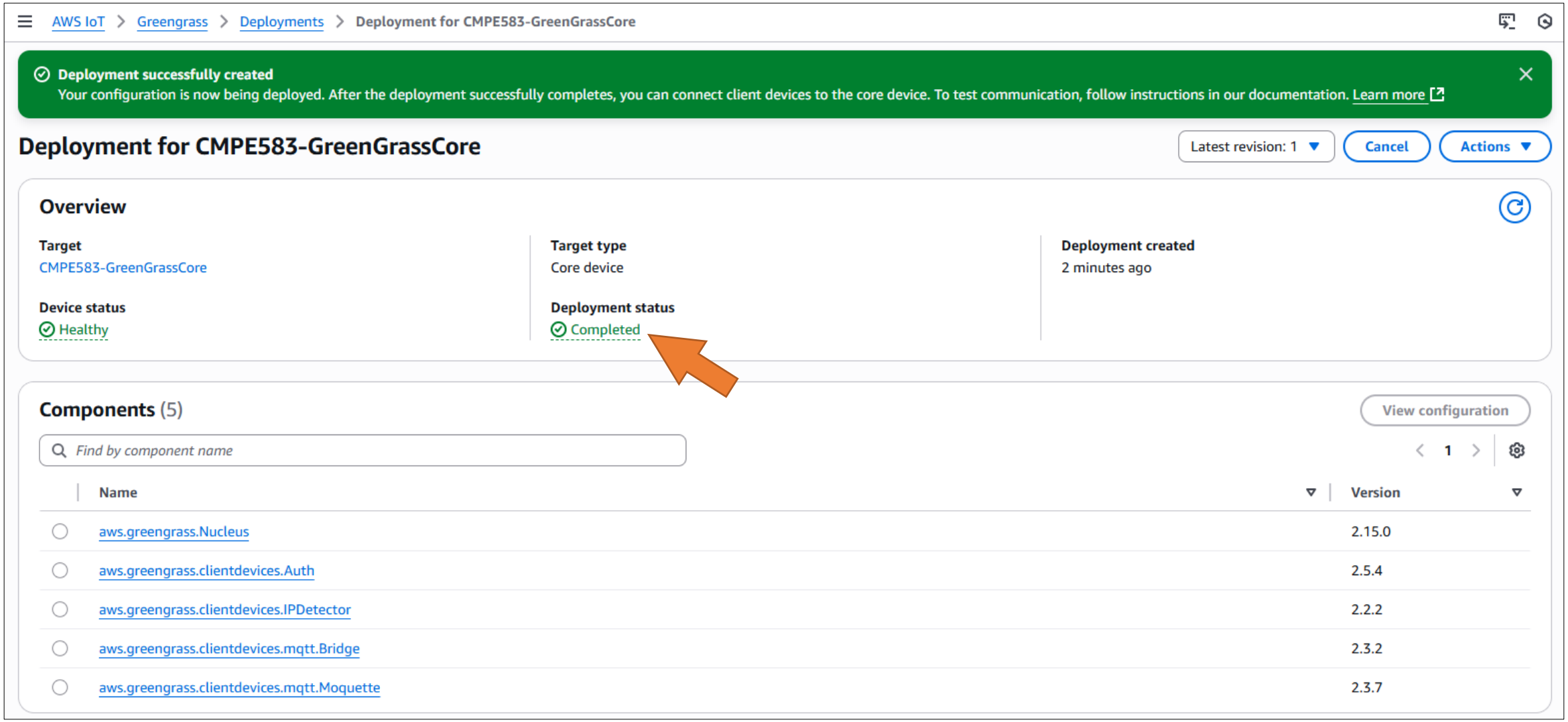
Step 3: Deploy Required Greengrass Components

Review & Deploy Greengrass Components



Step 3: Deploy Required Greengrass Components

Check Deployment Status



Onboarding Client Devices

Step 1: Install AWS IoT Device SDKs

Programming Languages

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

Step 1: Install AWS IoT Device SDKs

AWS IoT Device SDK v2 for Python

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

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

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

Install the AWS IoT Device SDK v2 for Python.

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

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

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

Step 2: Run Sample Device Discovery

With samples/basic_discovery.py

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

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

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

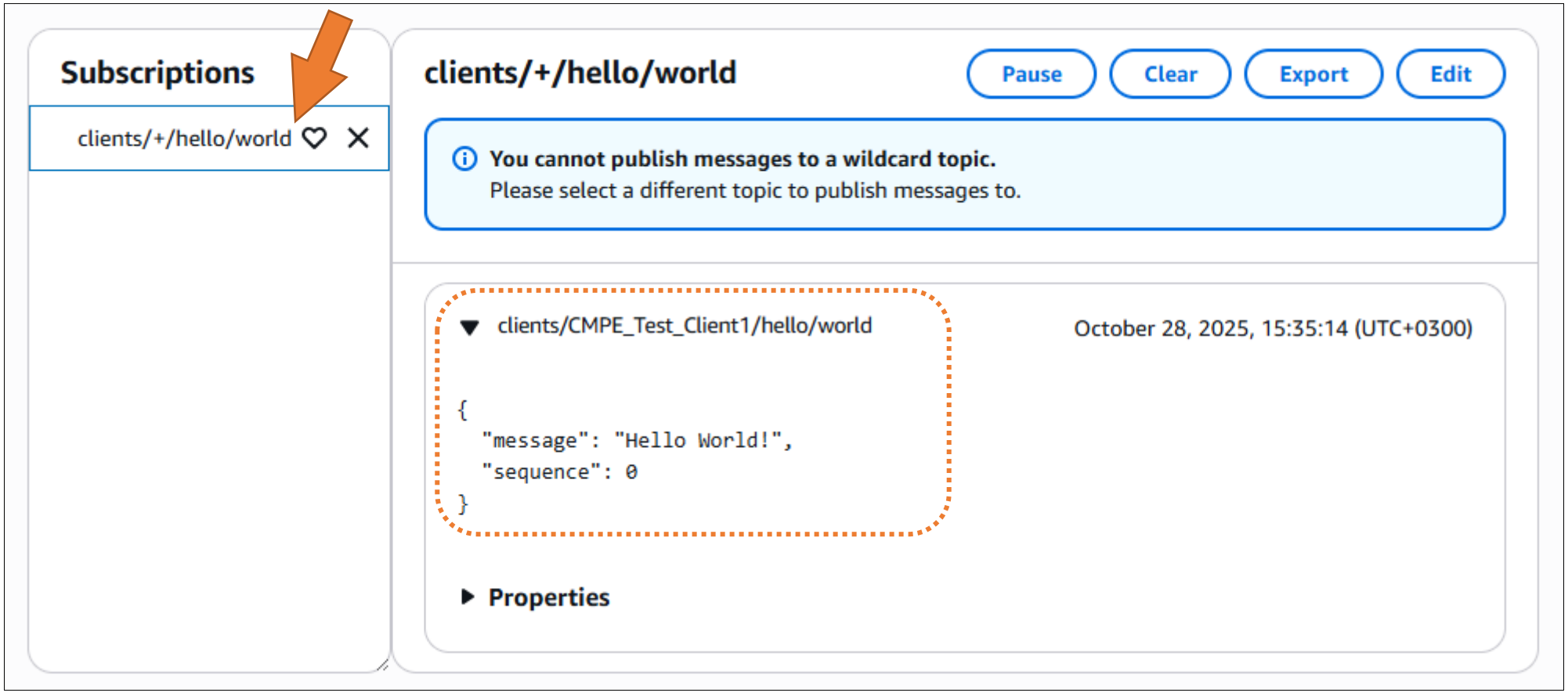
Step 2: Run Sample Device Discovery

Verify Client Device Connection From Terminal

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

Step 3: Verify Client Device Connection (MQTT Bridge)

Verify From MQTT Test Client



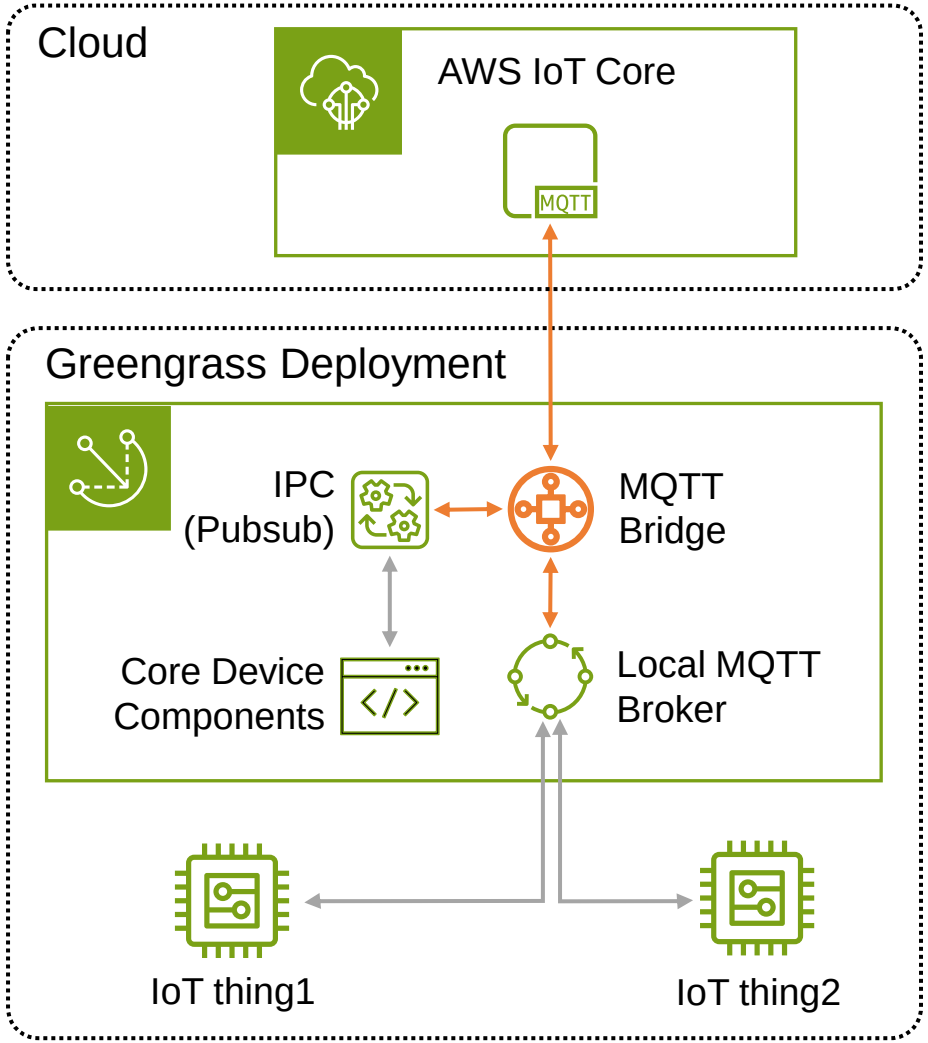
Communicate with Client Devices

MQTT Bridge: Understanding the Brokers

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

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

MQTT Bridge: Topic Mapping for Different Cases



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

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

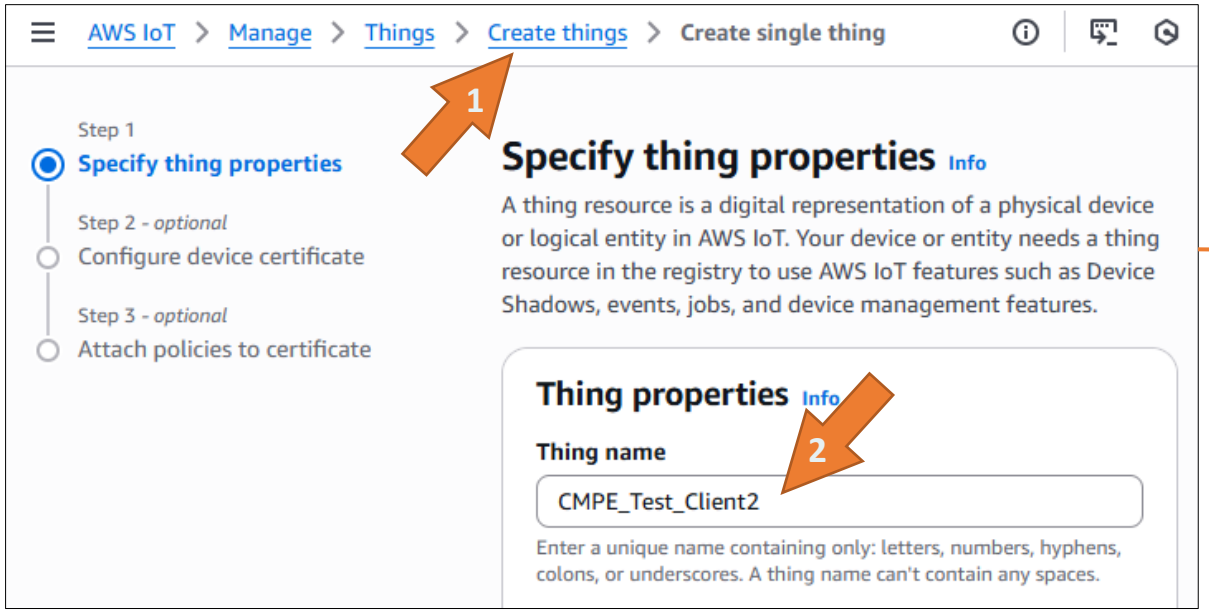
** Components can exchange messages directly via the Local IPC (Pubsub).

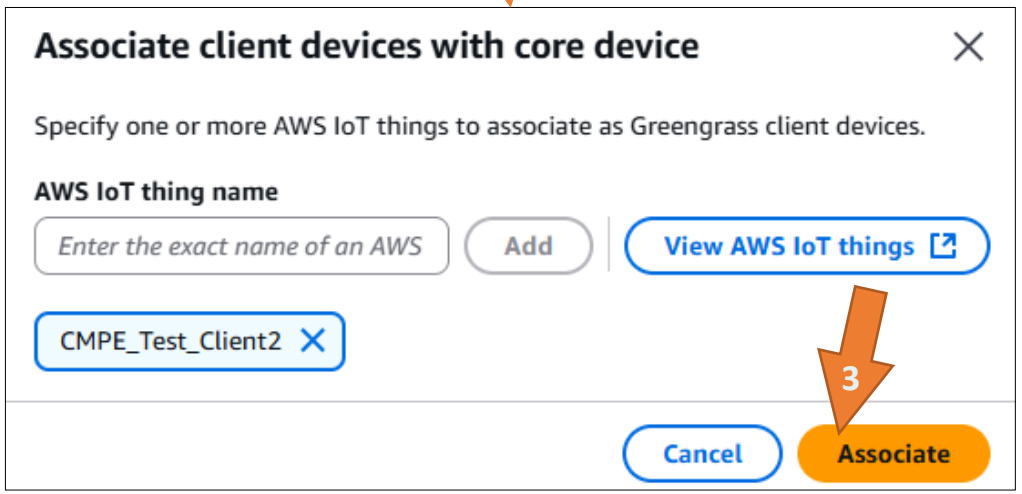
Steps for Client Device Communication

1. Create multiple IoT client devices
2. Configure cloud discovery to set up an MQTT bridge for topic mapping
3. Create custom component (with Python) to subscribe to local MQTT events
4. Deploy and review your component
5. Use the AWS IoT Device SDK to implement different use cases

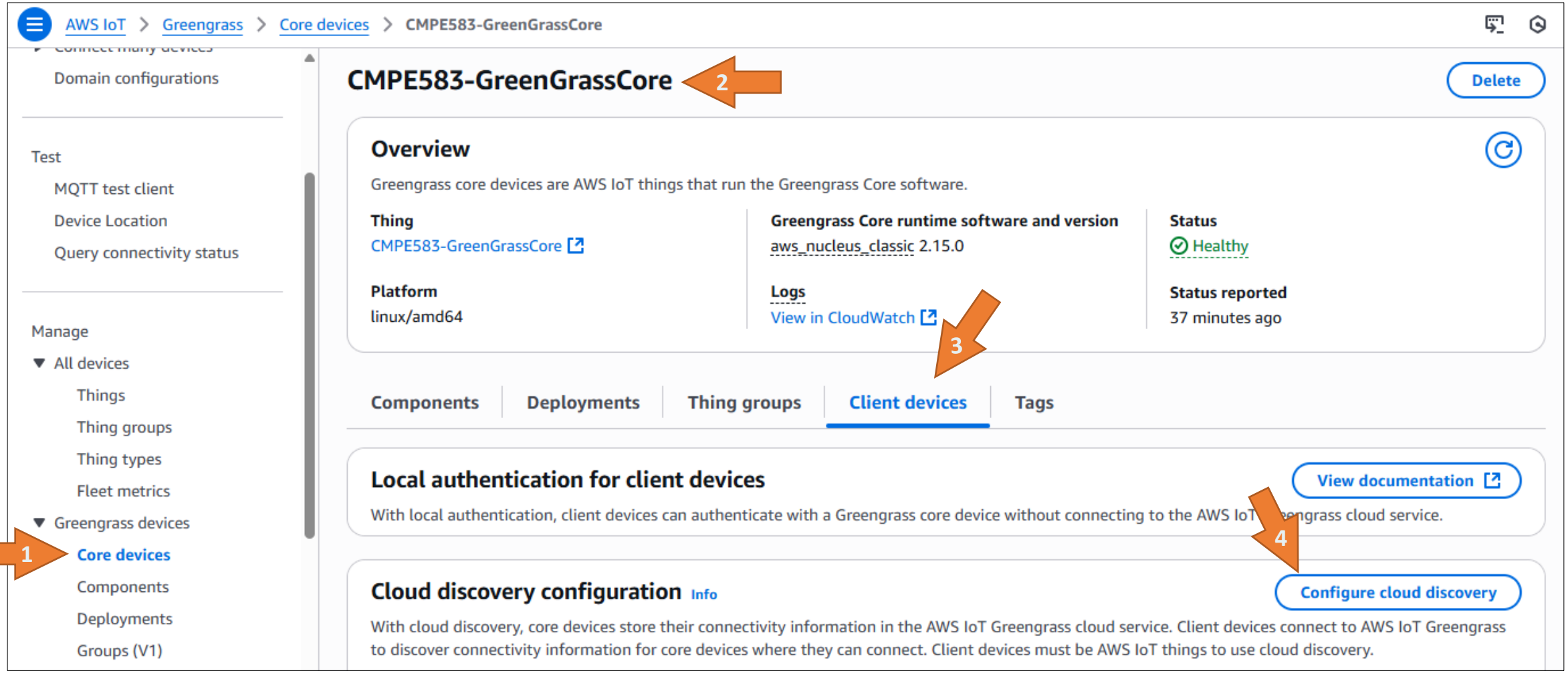
Step 1: Create More IoT Clients

Create Another Client and Associate It With Core Device



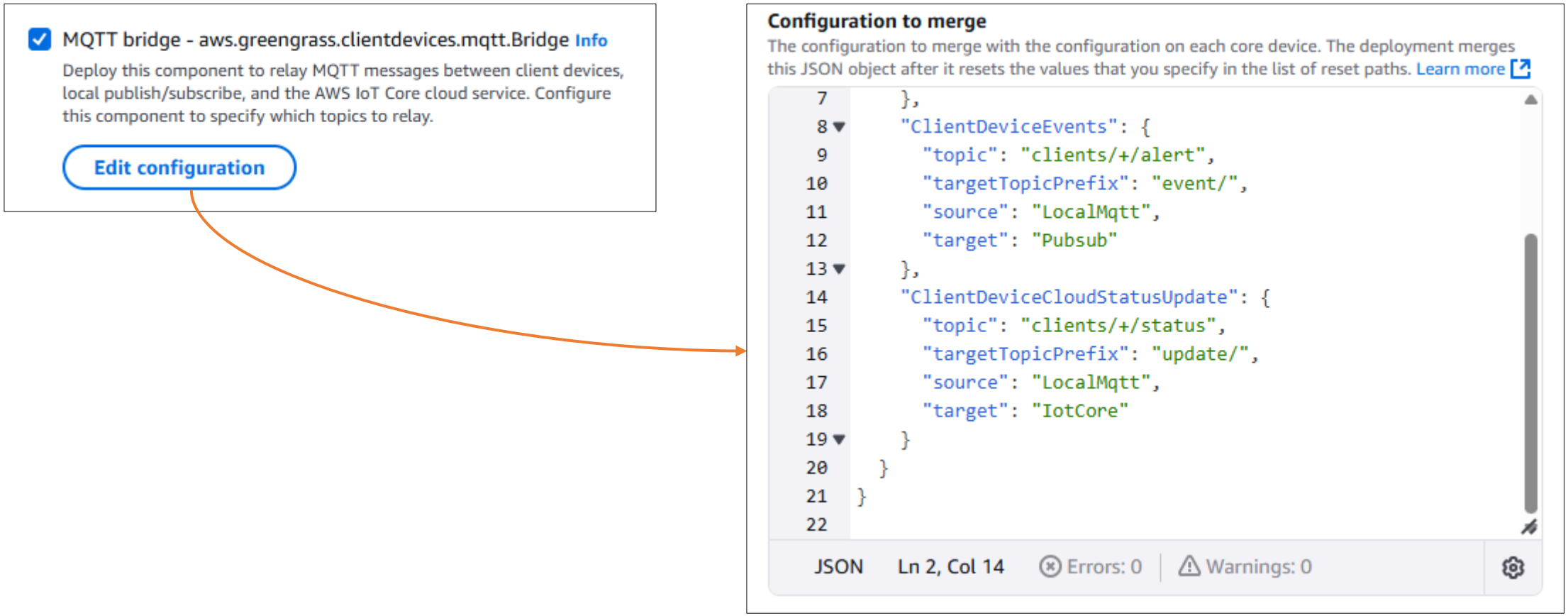


Step 2: Reconfigure Cloud Discovery



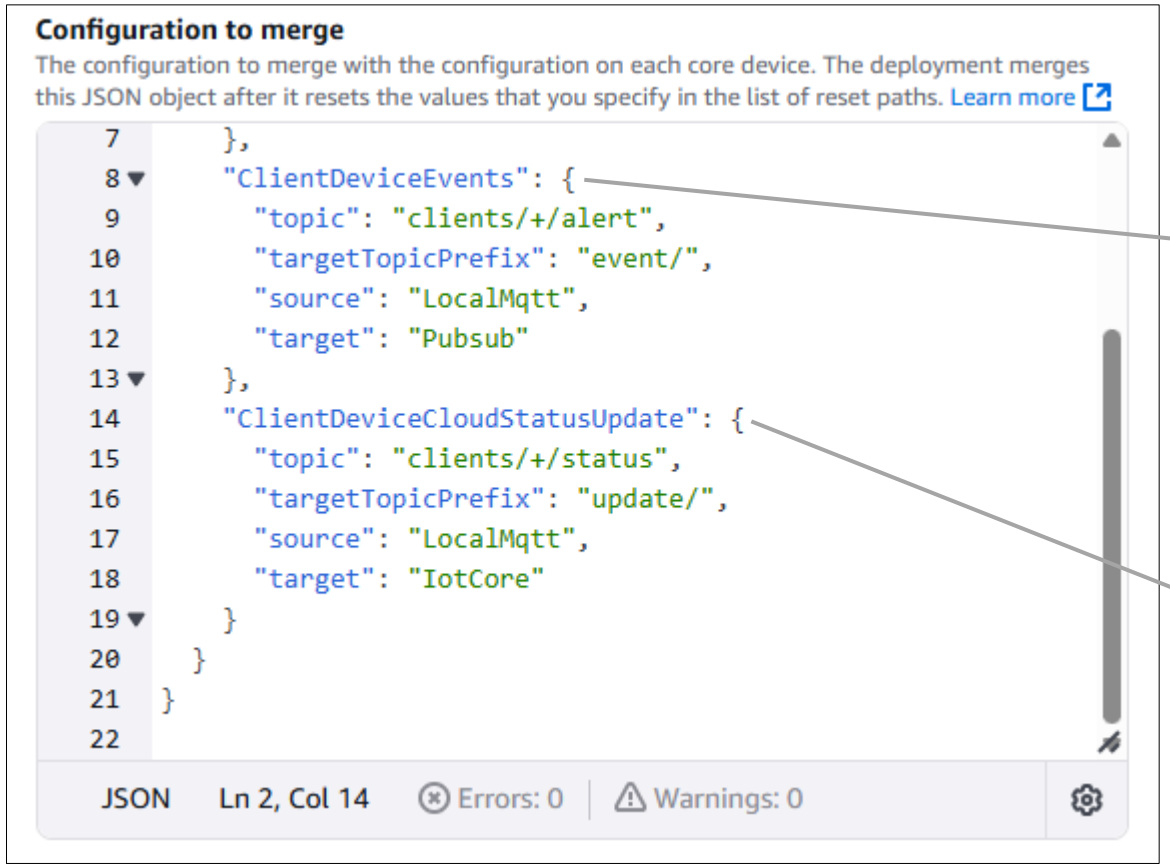
Step 2: Reconfigure Cloud Discovery

Reconfigure MQTT Bridge



Step 2: Reconfigure Cloud Discovery

New MQTT Topic Mapping

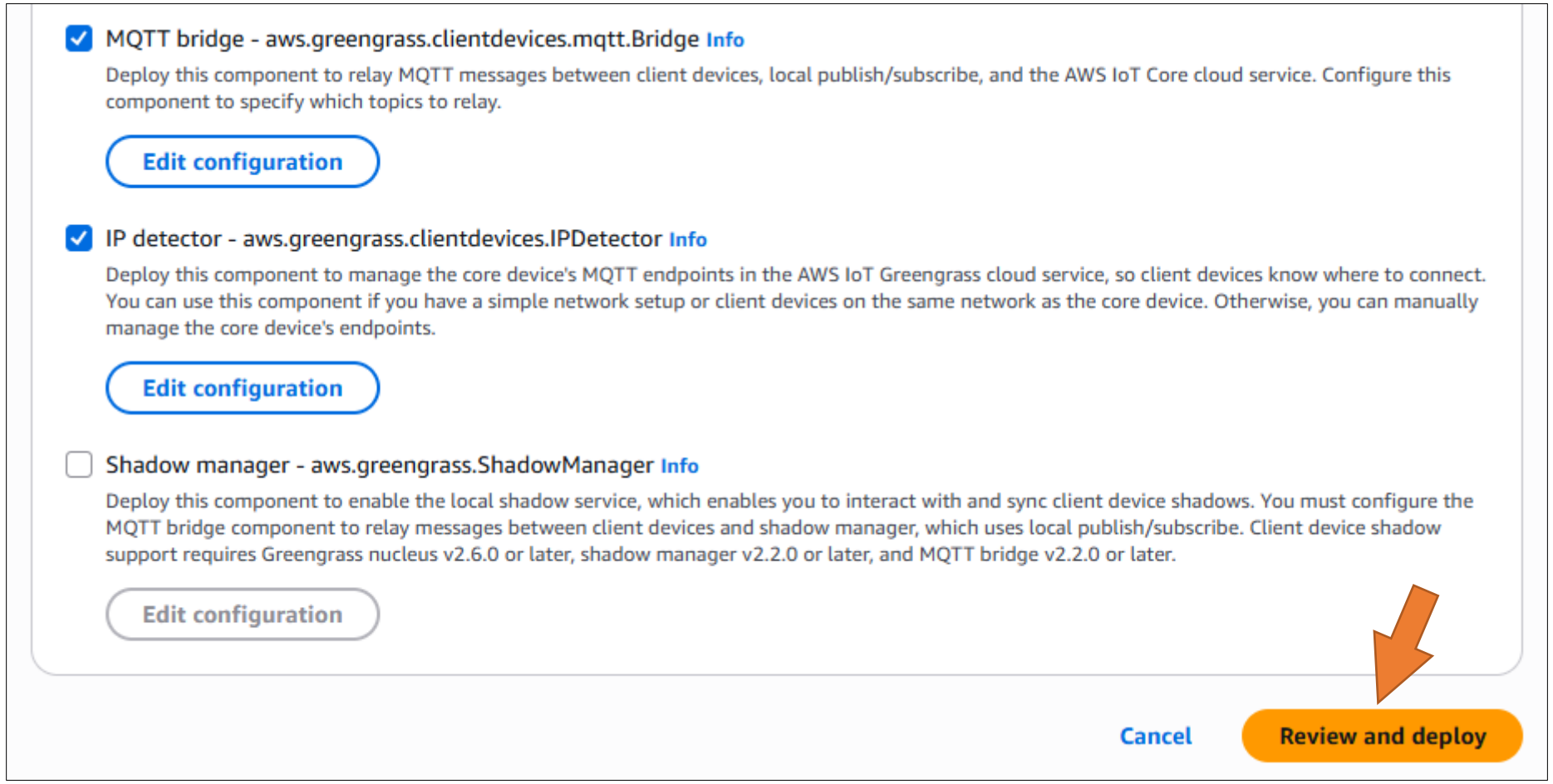


Relay messages from client devices to local IPC (Pubsub) through MQTT Bridge with **clients/+/alert** topic filter and add the *events/* prefix to the target topic. Core device components can receive this message with **events/clients/+/alert** topic.

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

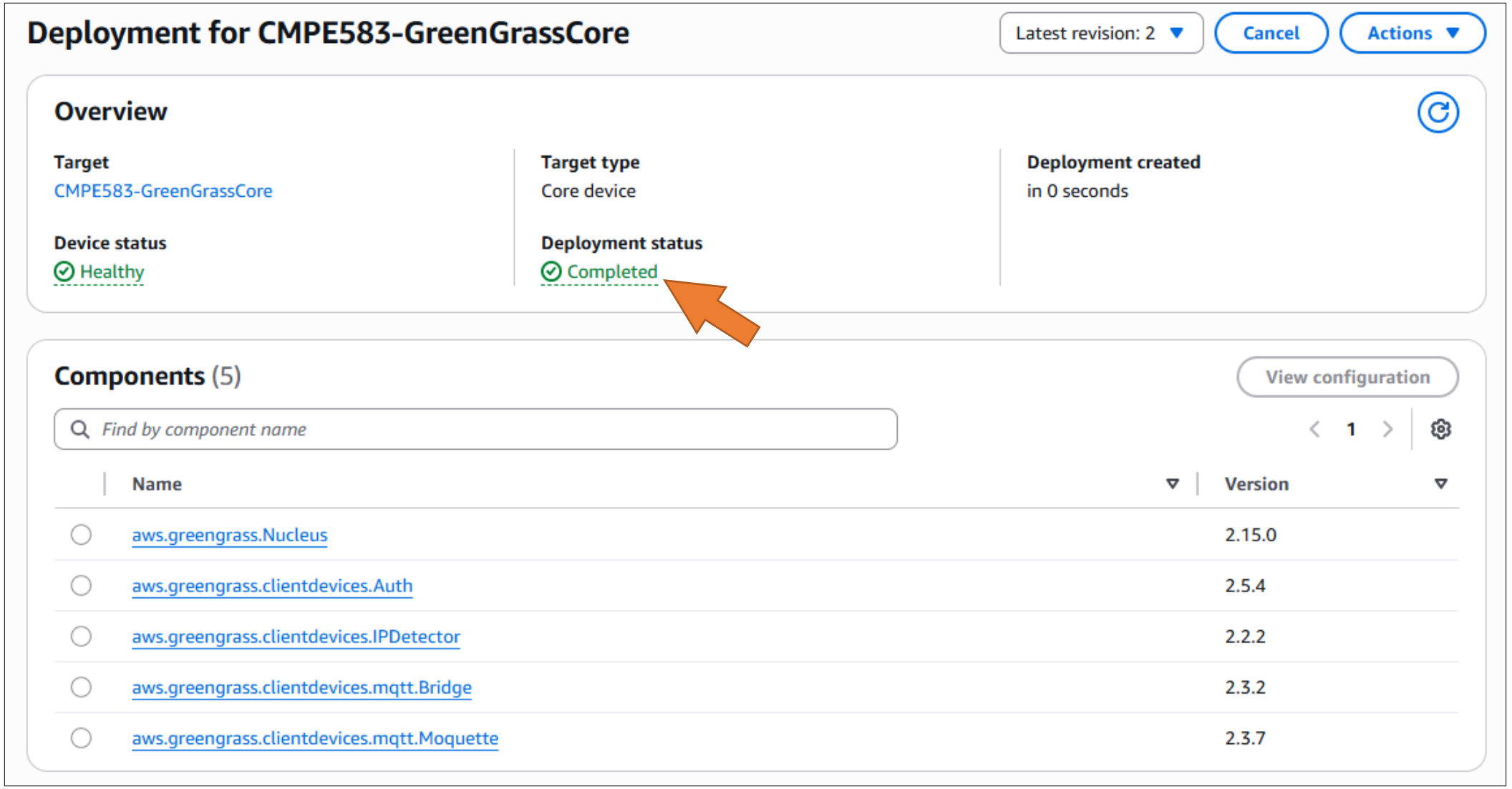
Step 2: Reconfigure Cloud Discovery

Review & Deploy the New Configuration



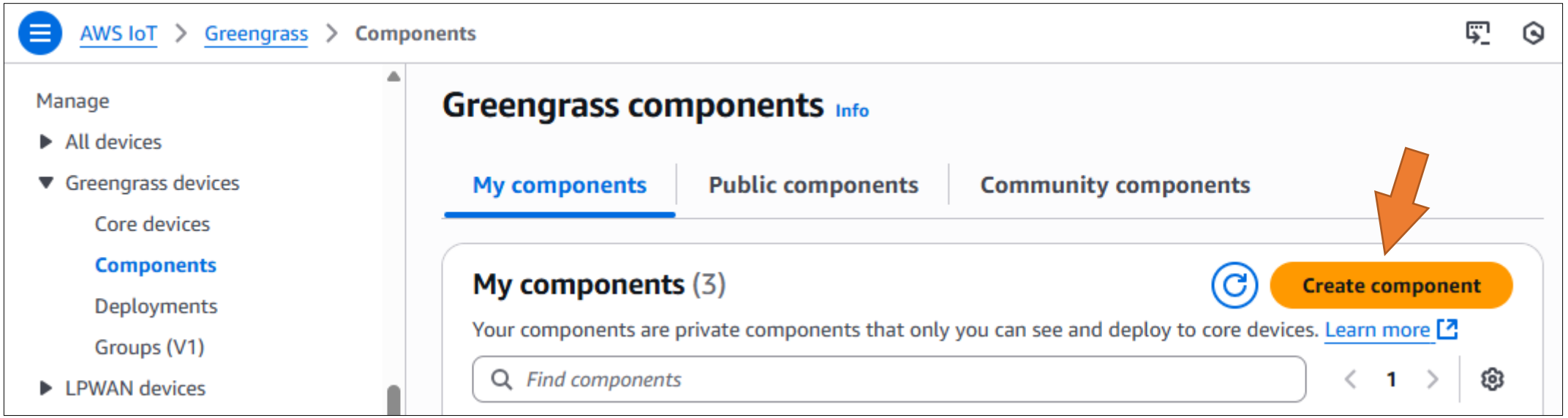
Step 2: Reconfigure Cloud Discovery

Check Deployment Status



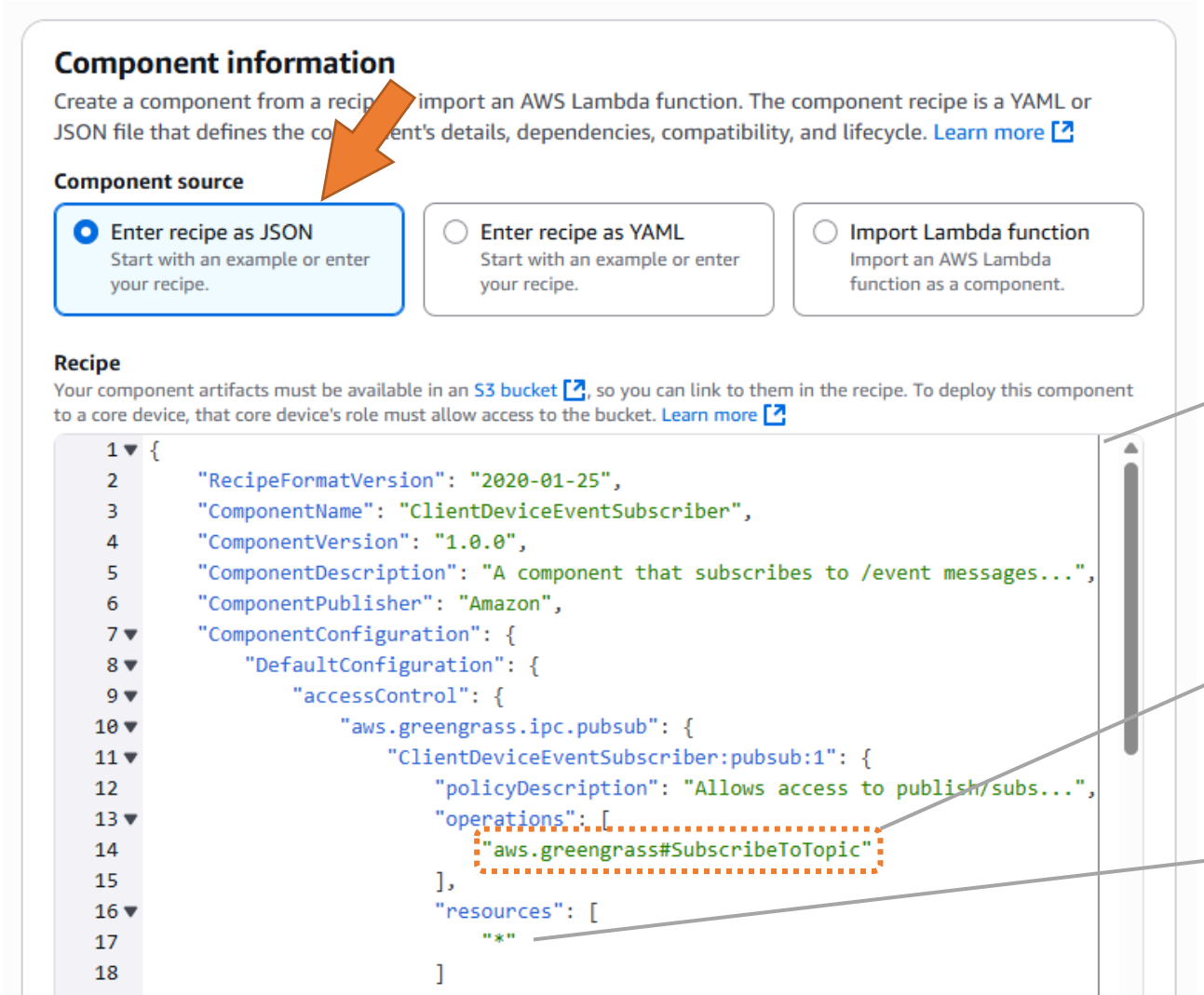
Step 3: Create a Custom Component

To Suscribe Local MQTT Events



Step 3: Create a Custom Component

Configure accessControl for Local Messages



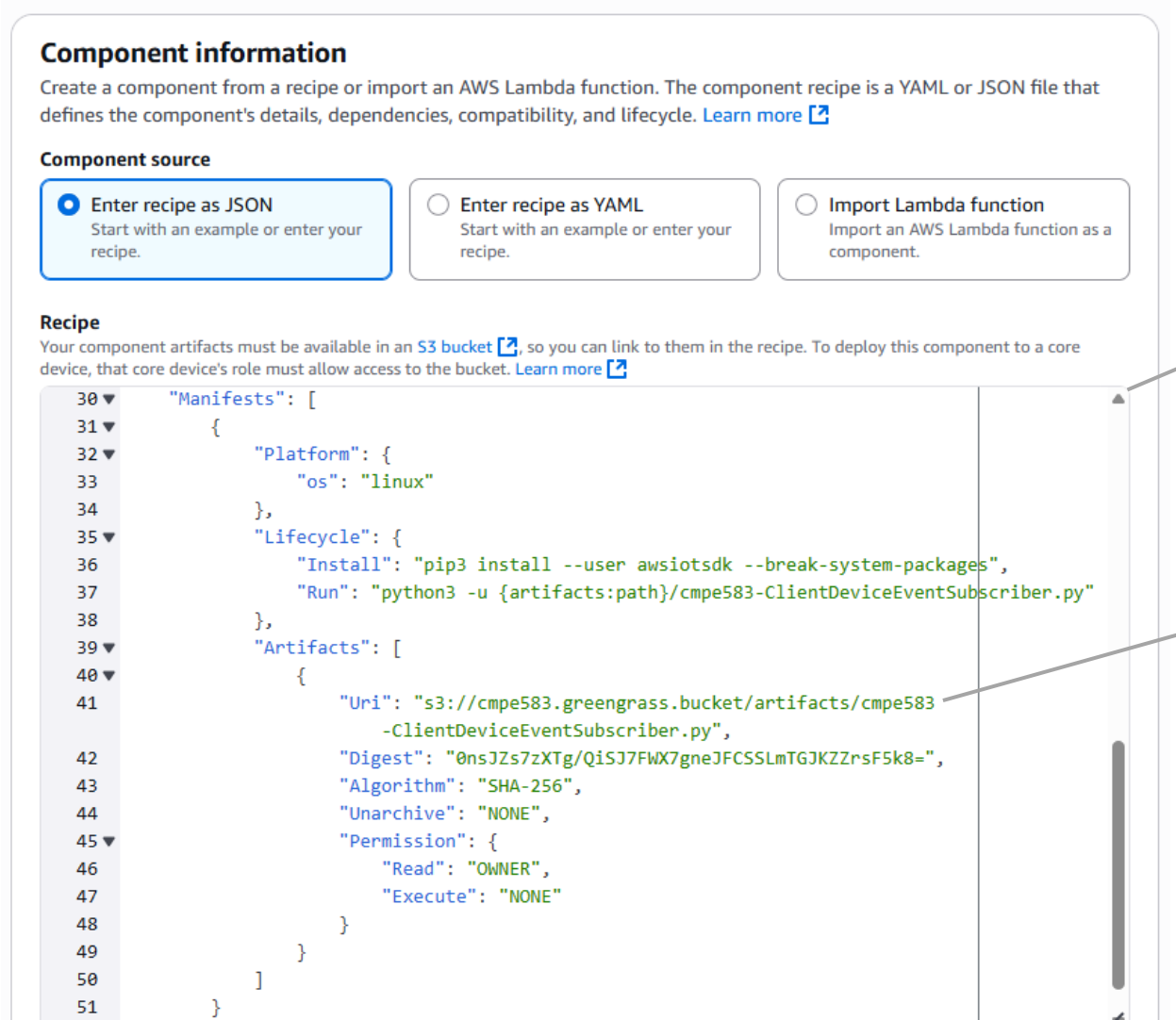
Check GitHub to see whole contents in ***cmpe583-ClientDeviceEventSubscriber.json***

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

Allow access to all topics.

Step 3: Create a Custom Component

Configure Artifacts in the Recipe



Component information

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

Component source

- Enter recipe as JSON — Start with an example or enter your recipe.
- Enter recipe as YAML — Start with an example or enter your recipe.
- Import Lambda function — Import an AWS Lambda function as a component.

Recipe

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

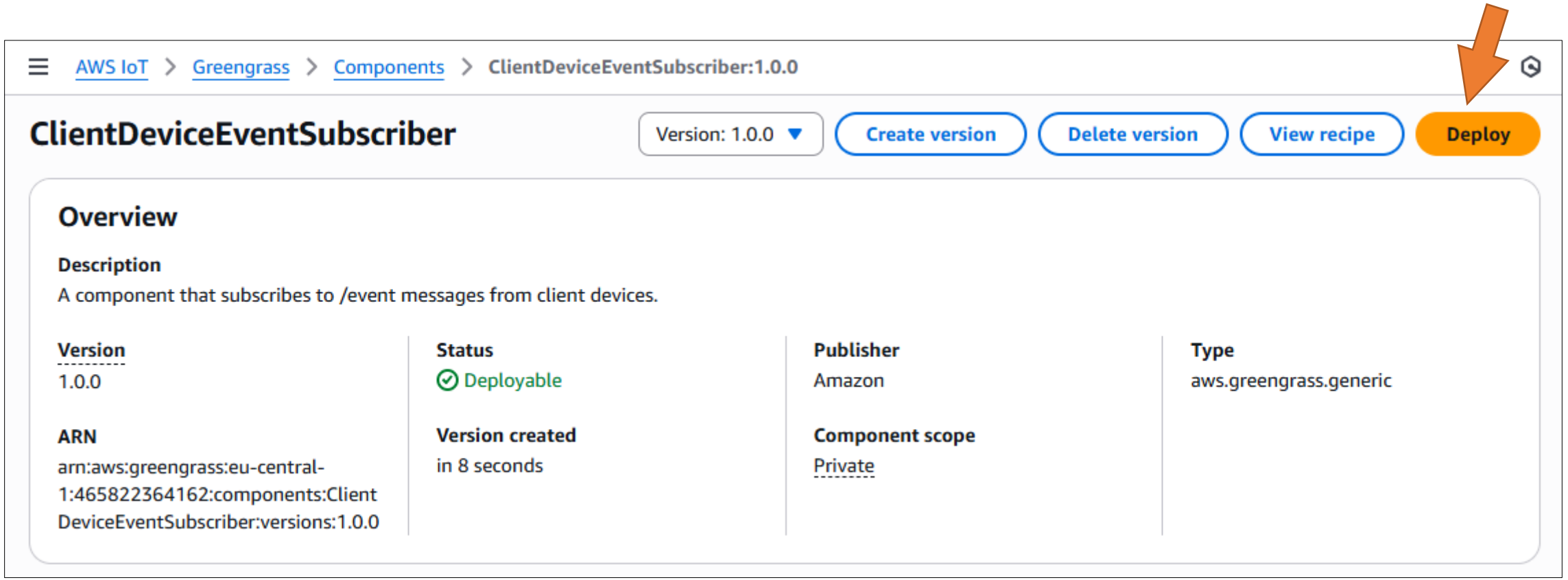
```
"Manifests": [
    {
        "Platform": {
            "os": "linux"
        },
        "Lifecycle": {
            "Install": "pip3 install --user awsiotsdk --break-system-packages",
            "Run": "python3 -u {artifacts:path}/cmpe583-ClientDeviceEventSubscriber.py"
        },
        "Artifacts": [
            {
                "Uri": "s3://cmpe583.greengrass.bucket/artifacts/cmpe583
                    -ClientDeviceEventSubscriber.py",
                "Digest": "0nsJZs7zXTg/QiSJ7FWX7gneJFCSSLmTGJKZZrsF5k8=",
                "Algorithm": "SHA-256",
                "Unarchive": "NONE",
                "Permission": {
                    "Read": "OWNER",
                    "Execute": "NONE"
                }
            }
        ]
    }
```

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.

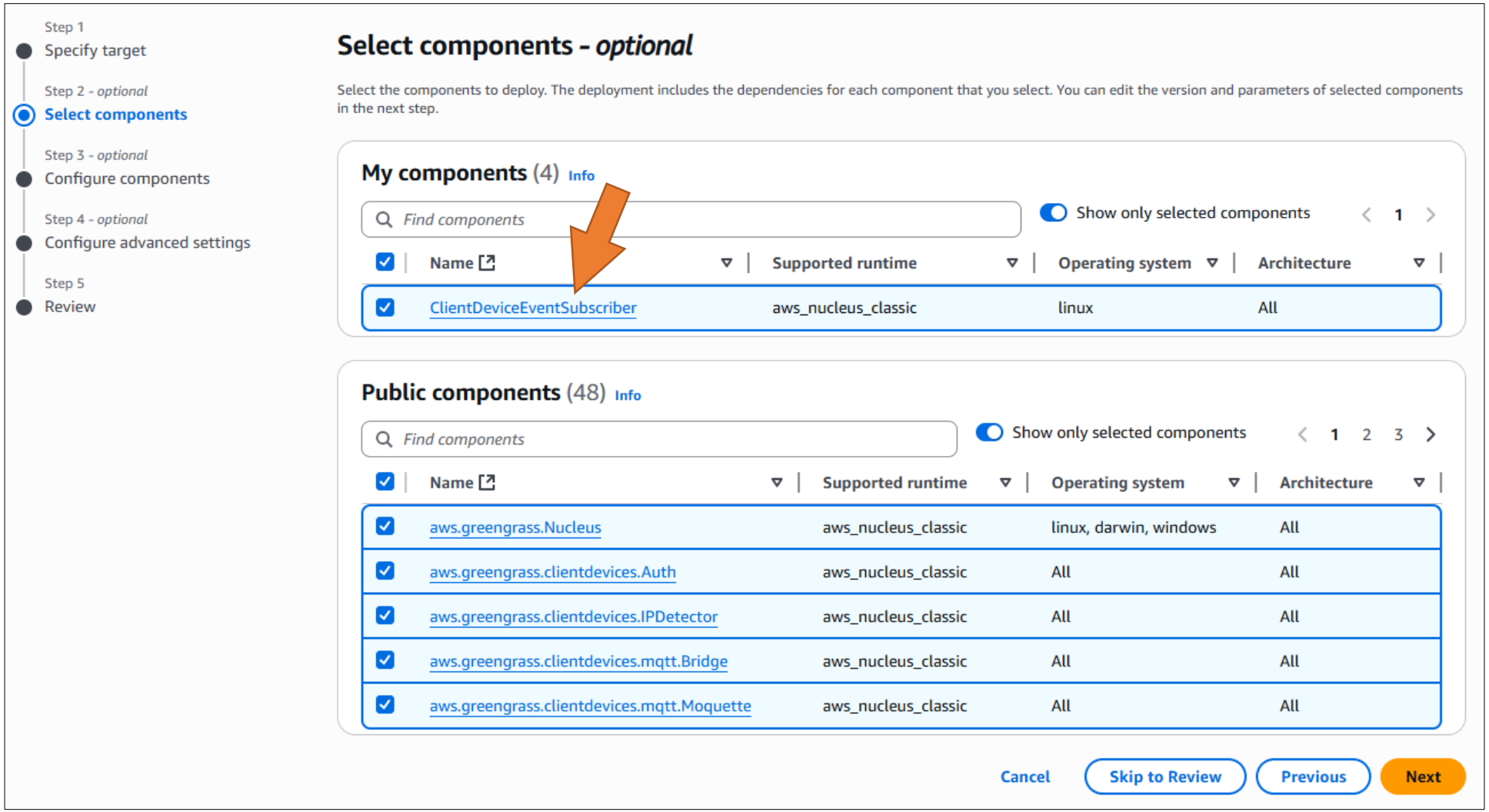
Step 4: Deploy Your Component

Via AWS IoT Greengrass (Console)



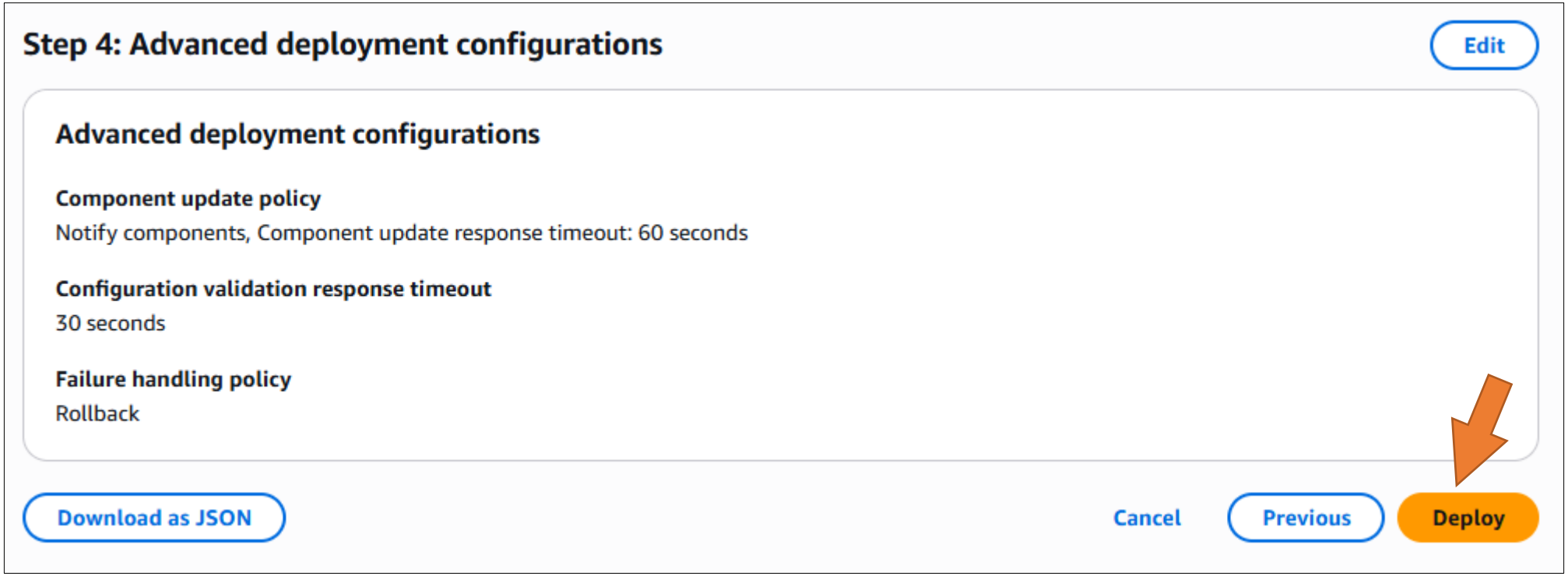
Step 4: Deploy Your Component

Select the Component



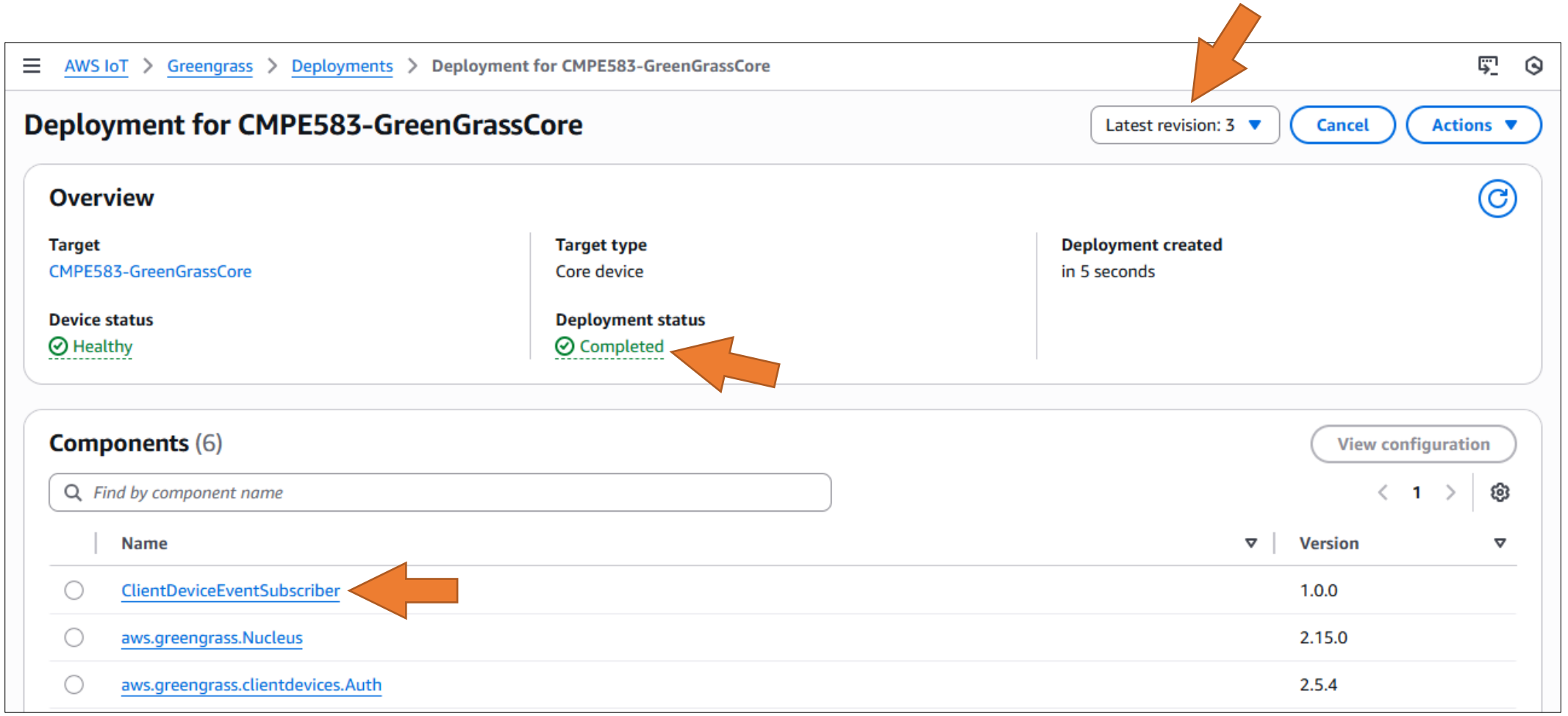
Step 4: Deploy Your Component

Review & Deploy



Step 4: Deploy Your Component

Check Deployment Status on AWS IoT Console



Step 4: Deploy Your Component

Check Deployment Status on Core Device

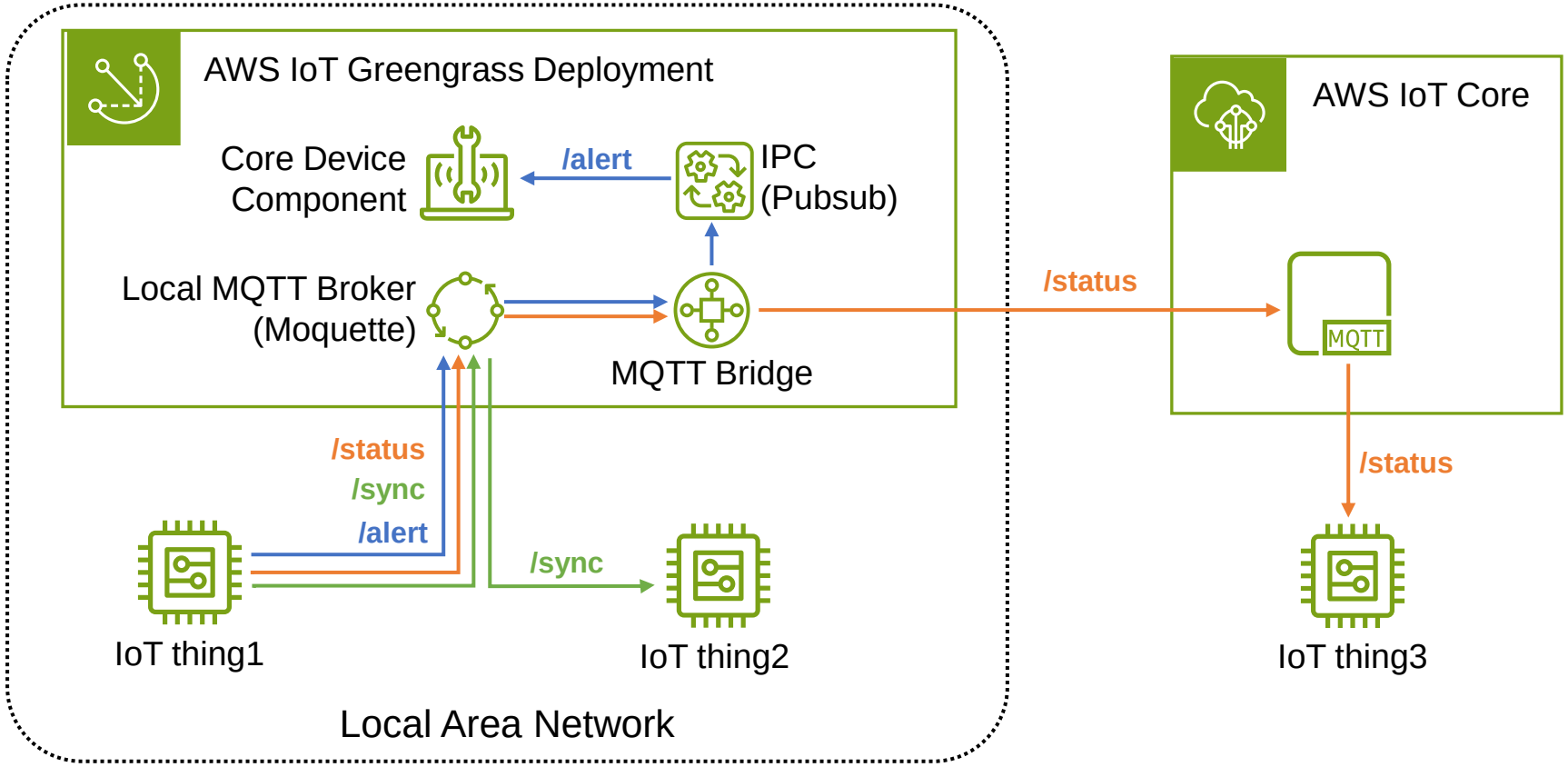
- Check ClientDeviceEventSubscriber.log file if the deployed component runs without error.
- If yes, you will see that the Core Device was subscribed to the topic successfully.

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

Four Different Device Communication Scenarios

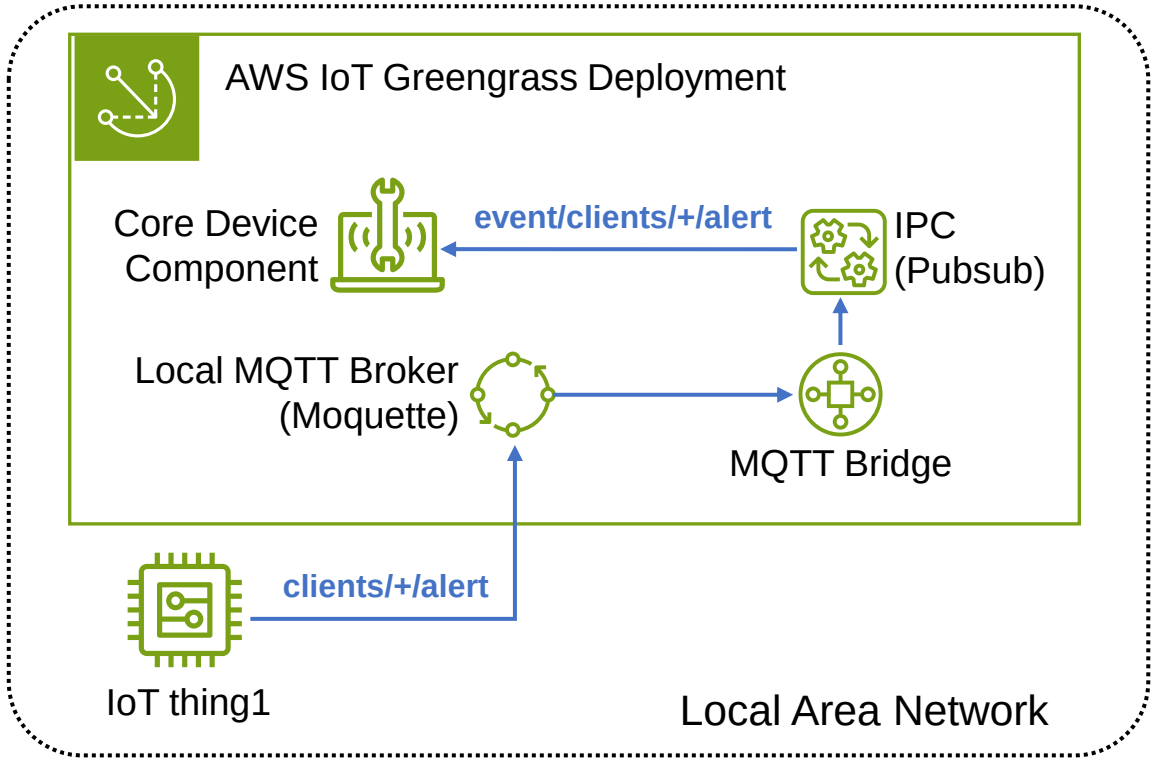
Communicating with Client Devices

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



Case 1: IoT Client to Core Device Communication

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



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

Publish Topic to Trigger Subscribers (Core Device)

- Publish the topic with basic_discovery.py script.

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

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

This message will be relayed to Core Device

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

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

Verify Process on Terminal

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

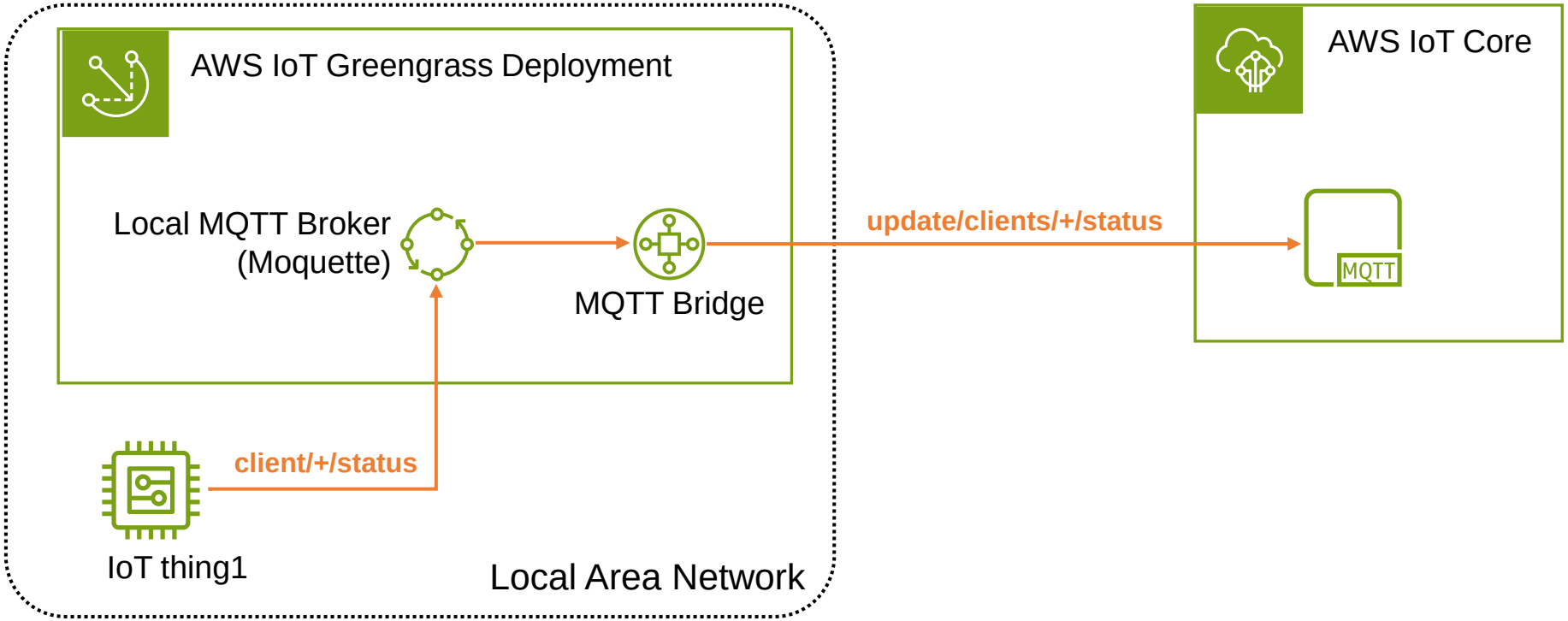
Step 2: Receive 'event/clients/+/alert' on Component

Verify Process on Core Device’s Terminal

```
root@cagatay:/home/cagatay#
root@cagatay:/home/cagatay# tail -n 100 /greengrass/v2/logs/ClientDeviceEventSubscriber.log
2025-10-28T13:23:57.755Z [INFO] (pool-3-thread-79) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services
.ClientDeviceEventSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW, command=["pip
3 install --user awsiotsdk --break-system-packages"]}
2025-10-28T13:23:59.243Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: awsiotsdk i
n /home/ggc_user/.local/lib/python3.12/site-packages (1.26.0). {scriptName=services.ClientDeviceEventSubscriber.lifecycl
e.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2025-10-28T13:23:59.243Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Requirement already satisfied: awscrt==0.2
8.1 in /home/ggc_user/.local/lib/python3.12/site-packages (from awsiotsdk) (0.28.1). {scriptName=services.ClientDeviceEv
entSubscriber.lifecycle.Install, serviceName=ClientDeviceEventSubscriber, currentState=NEW}
2025-10-28T13:23:59.743Z [INFO] (pool-3-thread-79) ClientDeviceEventSubscriber: shell-runner-start. {scriptName=services
.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=STARTING, command=["py
thon3 -u /greengrass/v2/packages/artifacts/ClientDeviceEventSubscriber/1.0.0..."]}
2025-10-28T13:24:00.110Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Successfully subscribed to topic: event/cl
ients/+/alert. {scriptName=services.ClientDeviceEventSubscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber,
currentState=RUNNING}
2025-10-28T13:58:19.828Z [INFO] (Copier) ClientDeviceEventSubscriber: stdout. Received new event from IoT Client: {"mess
age": "Production Anomaly Detected. Check the Production Plant!", "sequence": 0}. {scriptName=services.ClientDeviceEvent
Subscriber.lifecycle.Run, serviceName=ClientDeviceEventSubscriber, currentState=RUNNING}
root@cagatay:/home/cagatay#
```

Case 2: IoT Client to IoT Core Communication

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



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

Publish Topic to Trigger Subscribers (AWS IoT Core)

- Publish the topic with basic_discovery.py script.

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

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

This message will be relayed to AWS IoT Core

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

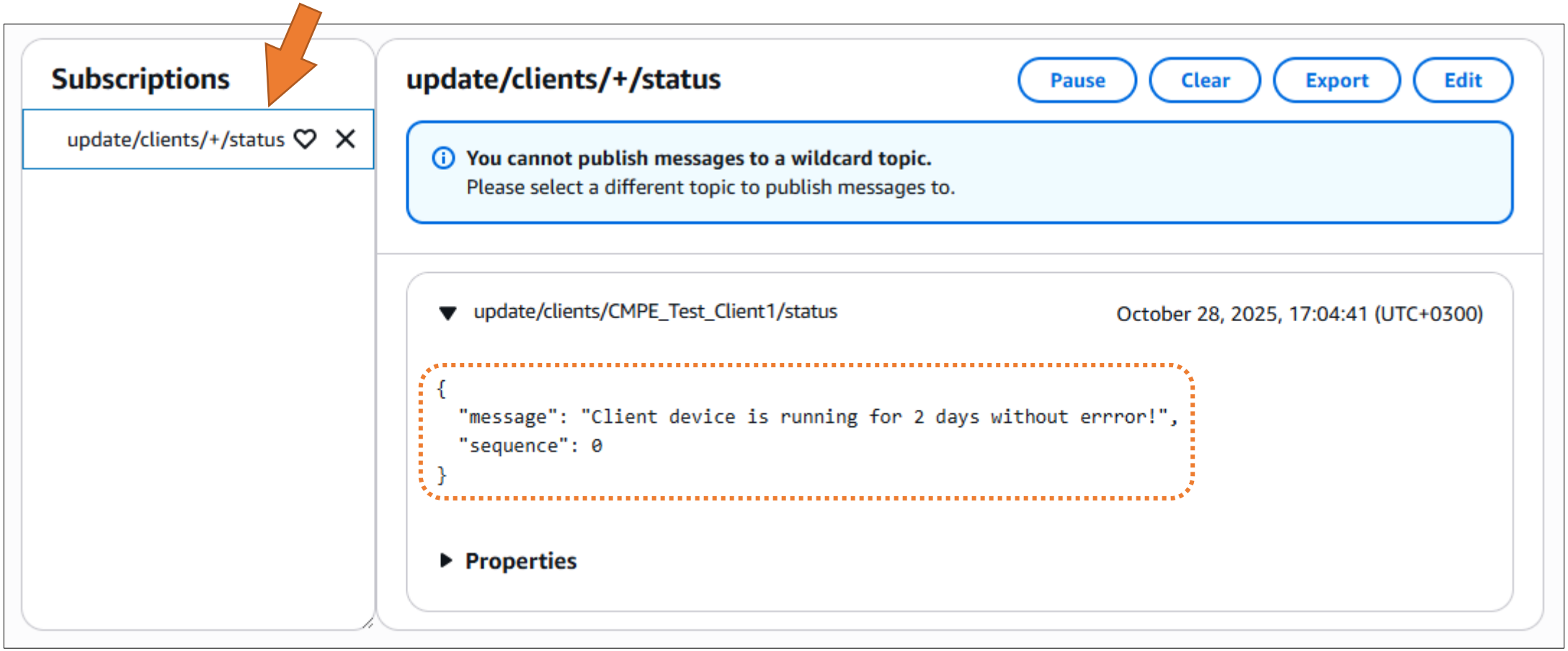
Step 1: Publish 'clients/+/status' Topic From IoT Client

Verify Process on Terminal

```
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$ python3 basic_discovery.py \
    --thing_name CMPE_Test_Client1 \
    --topic 'clients/CMPE_Test_Client1/status' \
    --message 'Client device is running for 2 days without errror!' \
    --ca_file /home/cagatay/shared_folder/greengrass/AmazonRootCA1.pem \
    --cert /home/cagatay/shared_folder/greengrass/certificate1.pem.crt \
    --key /home/cagatay/shared_folder/greengrass/private1.pem.key \
    --region eu-central-1 \
    --max_pub_ops 1 \
    --mode publish
Performing greengrass discovery...
Received a greengrass discovery result! Not showing result for possible data sensitivity.
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreenGrassCore at host 10.0.2.15 port 8883
Connected!
Successfully published to topic clients/CMPE_Test_Client1/status with payload `{"message": "Client device is running for 2 days without errror!"
, "sequence": 0}`
```

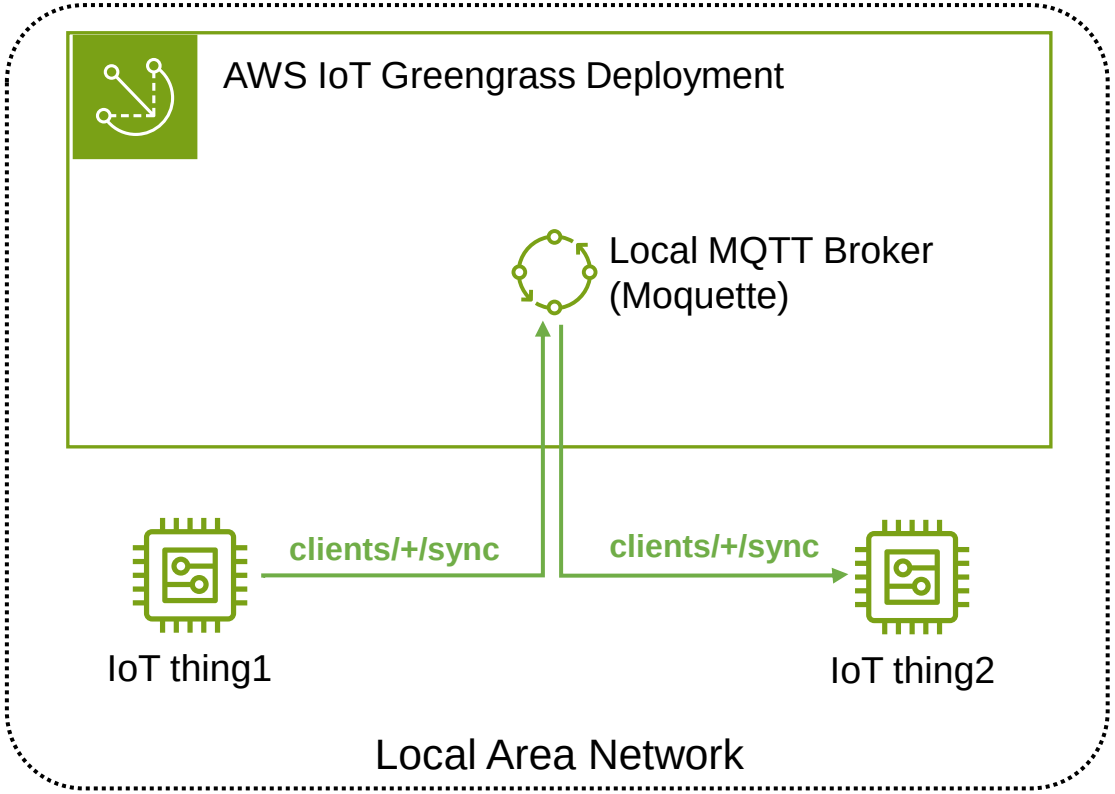
Step 2: Receive 'update/clients/+/status' on IoT Core

Verify Process on MQTT Test Client



Case 3: IoT Client to IoT Client Communication

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



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

Publish Topic to Trigger Subscribers (Test Client 2)

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

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

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

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

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

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

Verify Topic Published

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

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

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

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

Subscribe Topic From Test Client 2

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

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

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

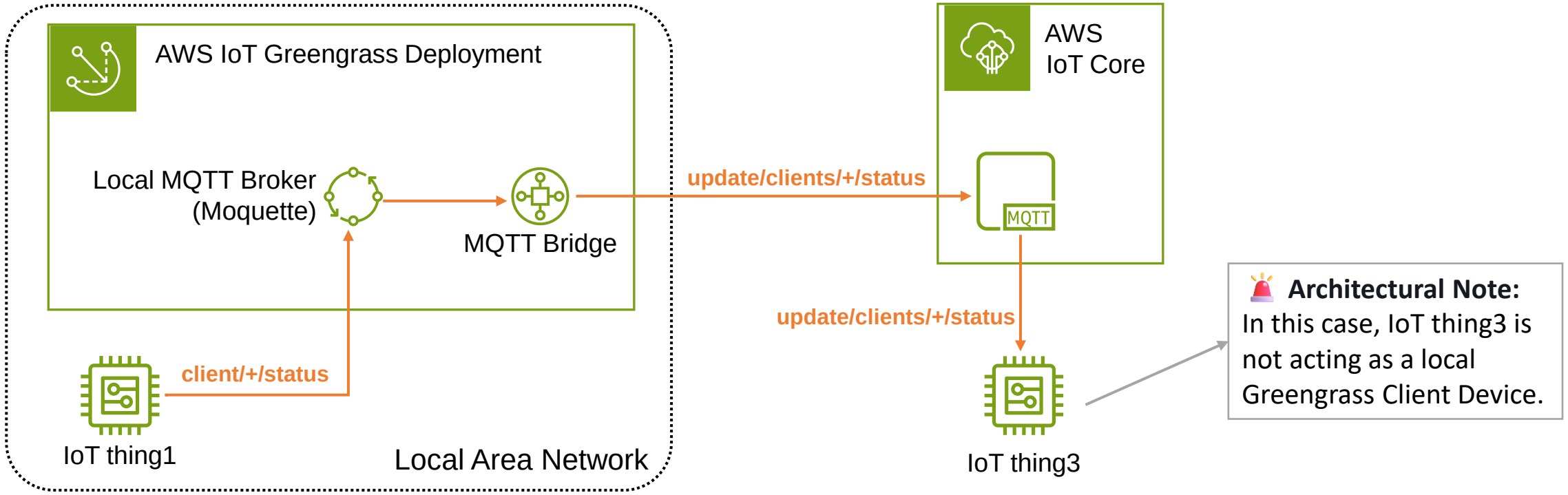
Step 2: Receive 'clients/+/sync' Topic on IoT Client2

Verify Topic Received

```
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$
(venvclient) cagatay@cagatay:~/Projects/aws-iot-device-sdk-python-v2/samples/greengrass$ python3 basic_discovery.py \
    --thing_name 'CMPE_Test_Client2' \
    --topic 'clients/+/sync' \
    --ca_file /home/cagatay/shared_folder/greengrass/AmazonRootCA1.pem \
    --cert /home/cagatay/shared_folder/greengrass/certificate2.pem.crt \
    --key /home/cagatay/shared_folder/greengrass/private2.pem.key \
    --region eu-central-1 \
    --mode subscribe
Performing greengrass discovery...
Received a greengrass discovery result! Not showing result for possible data sensitivity.
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreenGrassCore at host 10.0.2.15 port 8883
Connected!
Publish received on topic clients/CMPE_Test_Client1/sync
b'{"message": "Hello from IoT Client 1!", "sequence": 0}'
```

Case 4: IoT Core to IoT Client Communication

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



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

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

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

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

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

This message will be relayed to AWS IoT Core


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

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

Verify Topic Published

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

Performing greengrass discovery...
Received a greengrass discovery result! Not showing result for possible data sensitivity.
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreenGrassCore at host 10.0.2.15 port 8883
Connected!
Successfully published to topic clients/CMPE_Test_Client1/status with payload `{"message": "Client device is running for 2 days
 without errror!", "sequence": 0}`
```

Receiving Events From AWS IoT Core

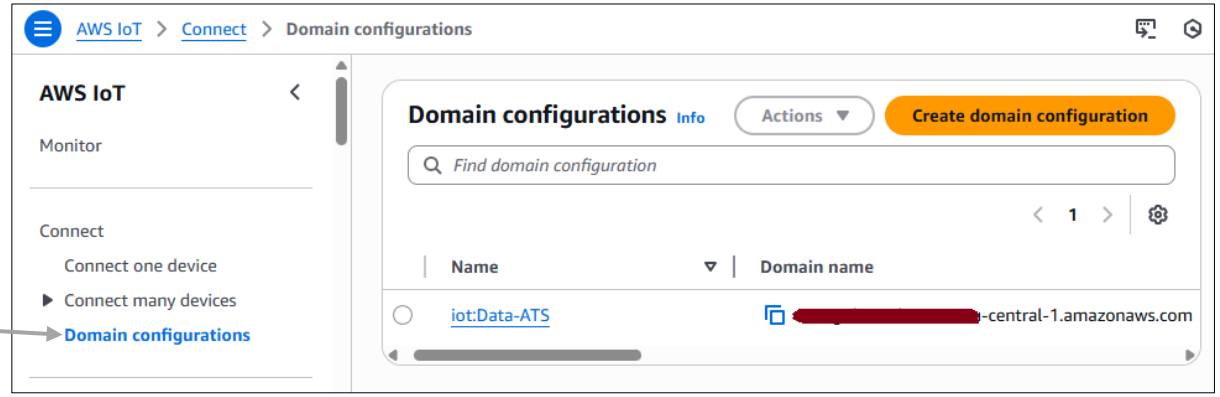
- Subscribe events published from AWS IoT Cloud with pubsub.py script using an empty --message.
- Don’t forget to modify MQTT Bridge for topic mapping!

✓ Use an empty--message with pubsub.py script to subsribe and receive events on IoT client device.

```
(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
```

✓ Your account’s endpoint can be found under the domain configuration settings



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

Subscribe Topic From Test Client 3

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

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

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

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

Verify Topic Received

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

Starting MQTT5 X509 PubSub Sample

==== Creating MQTT5 Client ====

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

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

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

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

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

Troubleshooting

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

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

Thank You