Edge Computing with AWS IoT Greengrass

Çağatay Sönmez

1 November 2024

The Primary Sources Used in This Lecture

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

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

⚠ Reminder

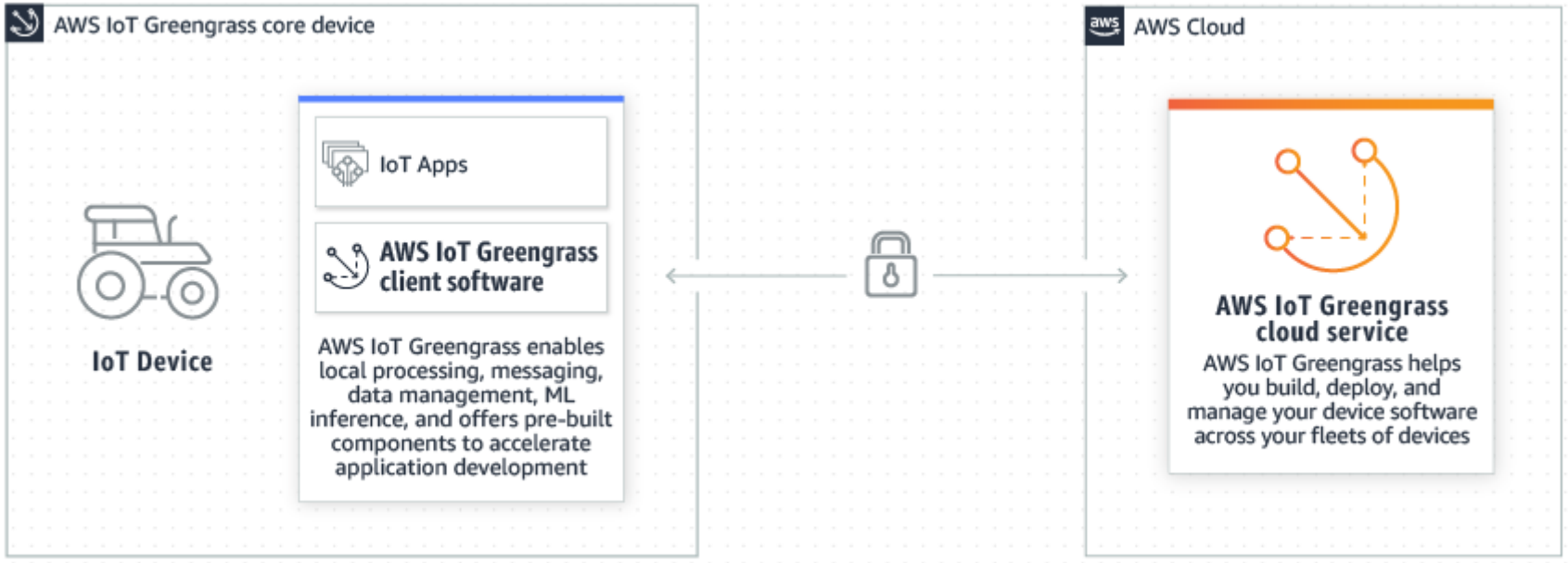
Monitor AWS **Free Tier** usage via **Budget Alerts** to avoid unexpected charges!

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

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

What is AWS IoT Greengrass?

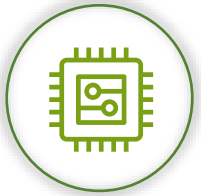
- AWS IoT Greengrass is software that extends cloud capabilities to local devices. This enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. Local devices can also communicate securely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



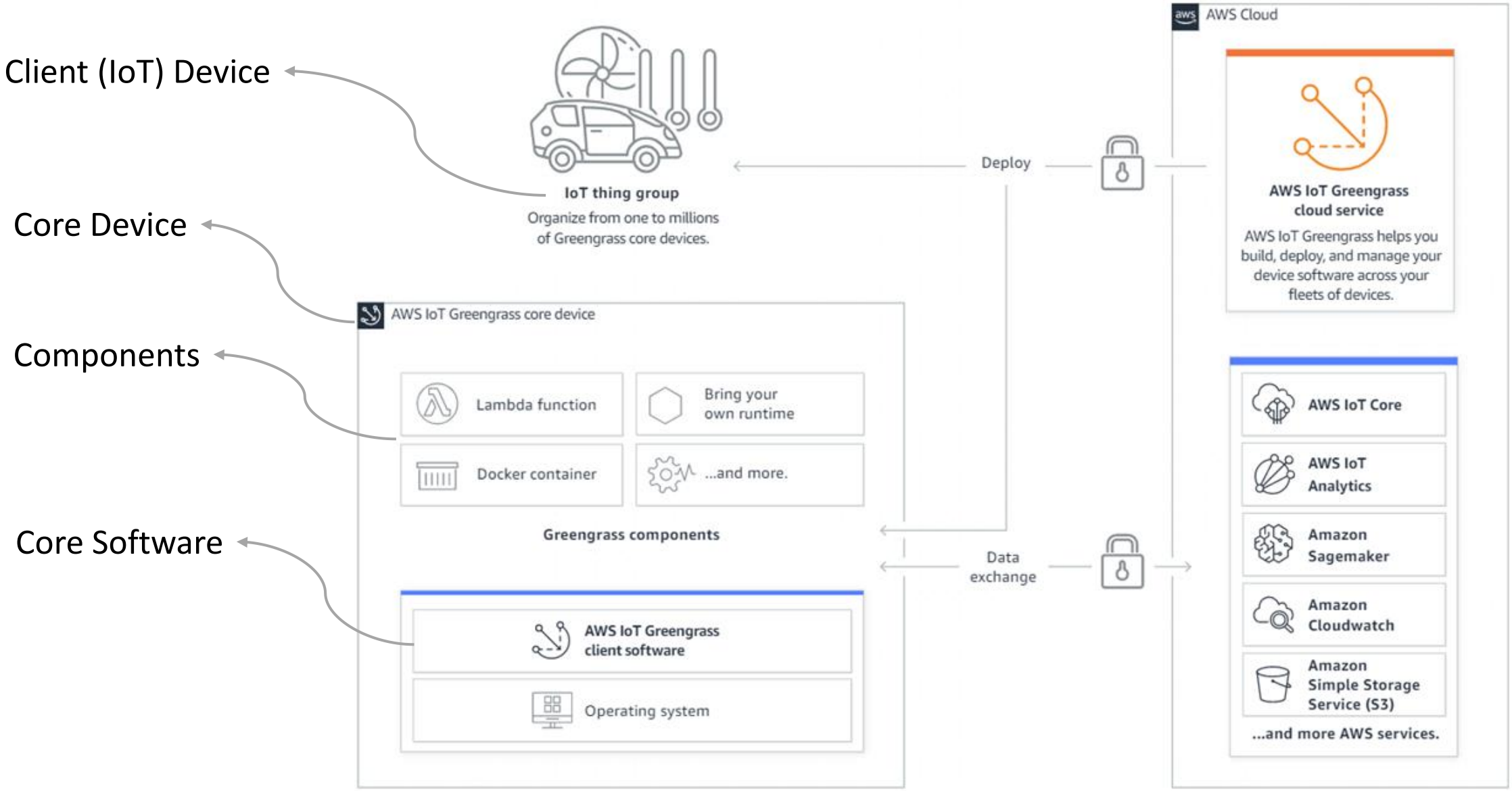
Greengrass Client Device: Any device (e.g. an industrial sensor) that communicates with a Greengrass Core device over MQTT. Client devices, which could be an IoT device, a mobile app, or any other system, use the *AWS IoT Device SDK* to interact with Greengrass Core devices.

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

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

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

How AWS IoT Greengrass works



Selecting Client's Operating System

- Some features are supported on only certain operating systems!

Security

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 IoT secure tunneling	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
Publish data to Amazon Kinesis Data Firehose delivery streams using the Kinesis Data Firehose component	Yes	No
Publish video streams to Amazon Kinesis Video Streams using the edge connector for Kinesis Video Streams component	Yes	No

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

AWS IoT Greengrass Core Software

AWS IoT Greengrass Core Software

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

Step 1: Set Up Your Environment

For Linux Based Devices

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

For Debian-based or Ubuntu-based distributions:

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

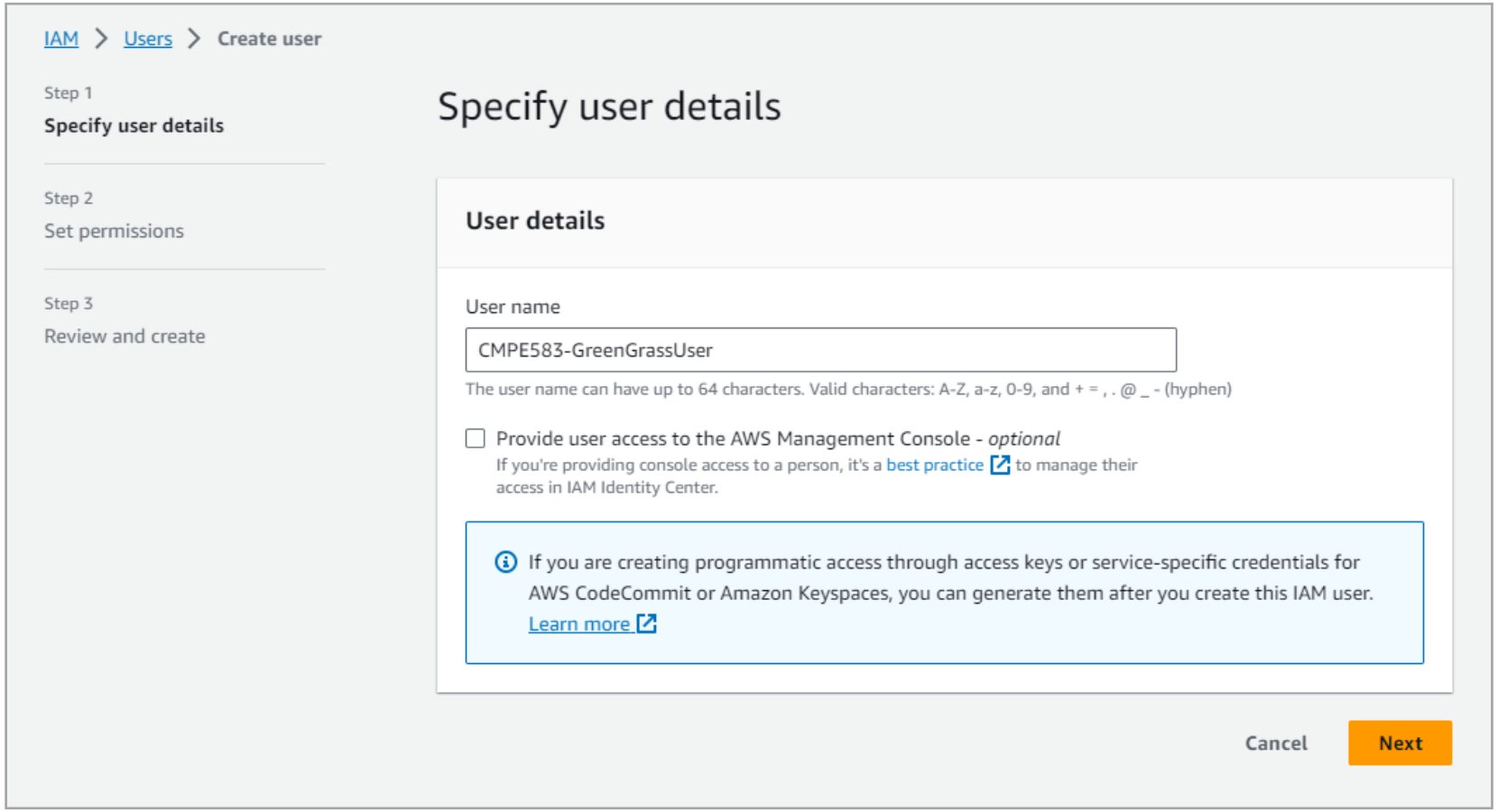
For Red Hat-based distributions:

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

Step 2: Create AWS Credentials

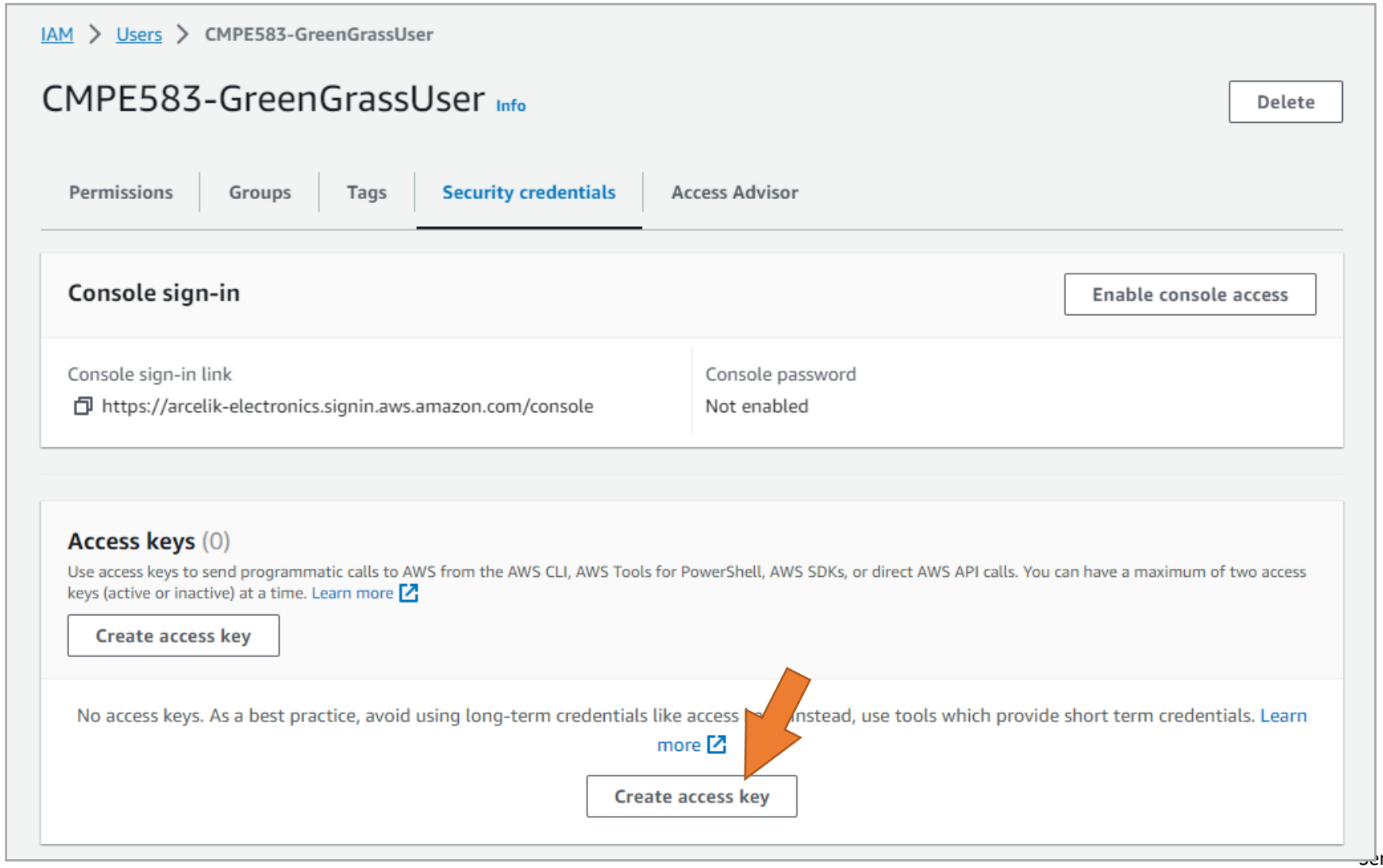
Create a Temporary User

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



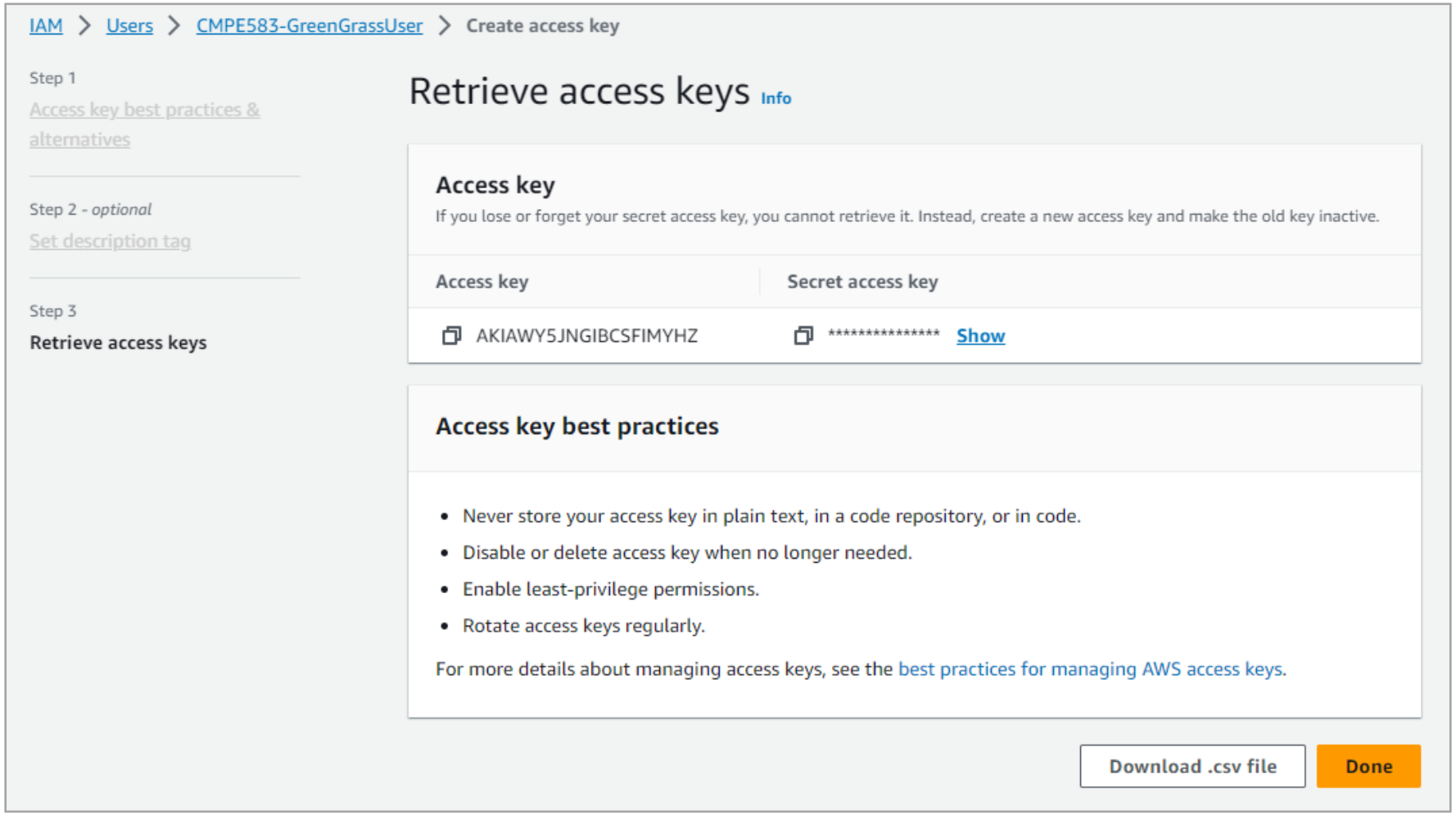
Step 2: Create AWS Credentials

Create Access Keys for the User

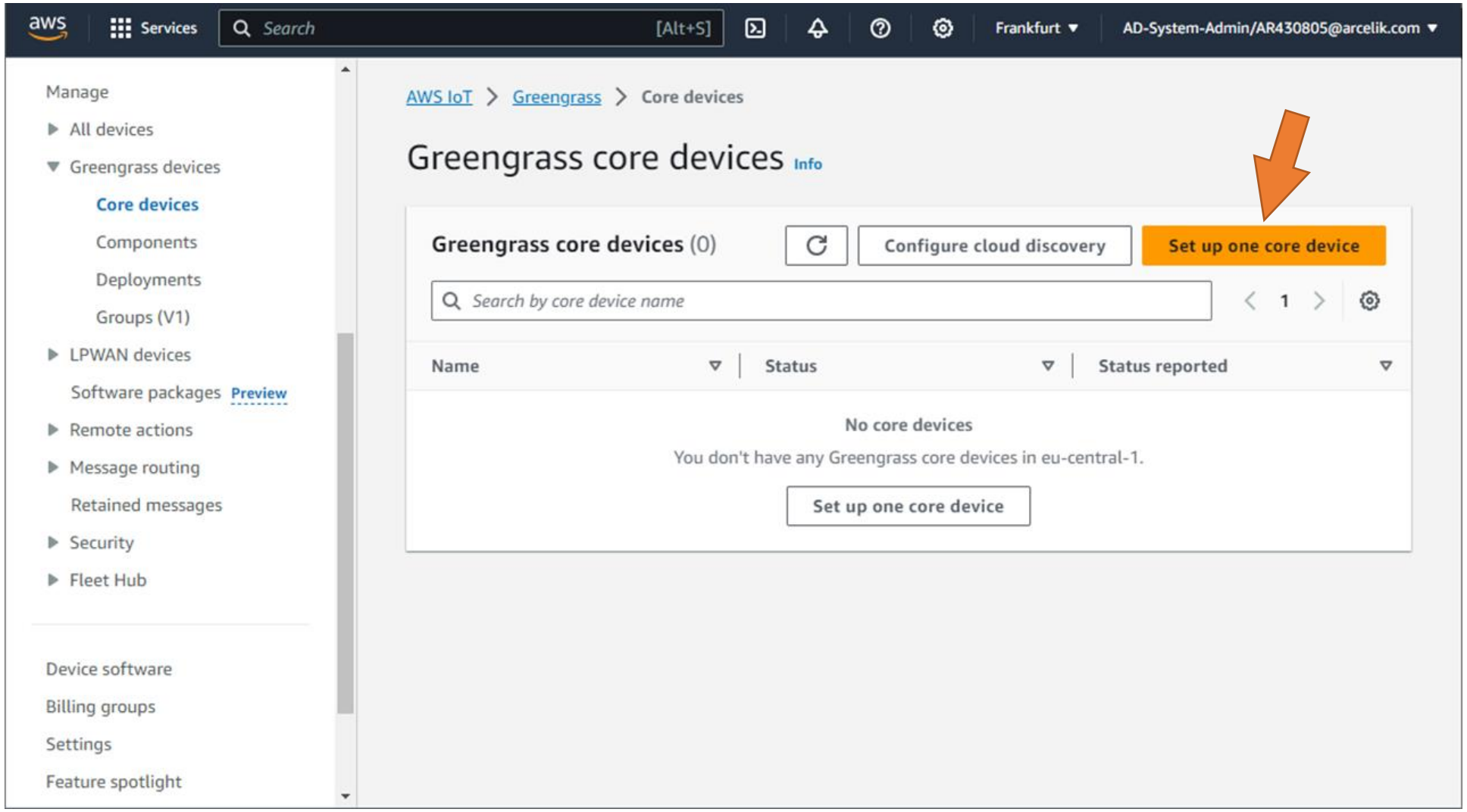


Step 2: Create AWS Credentials

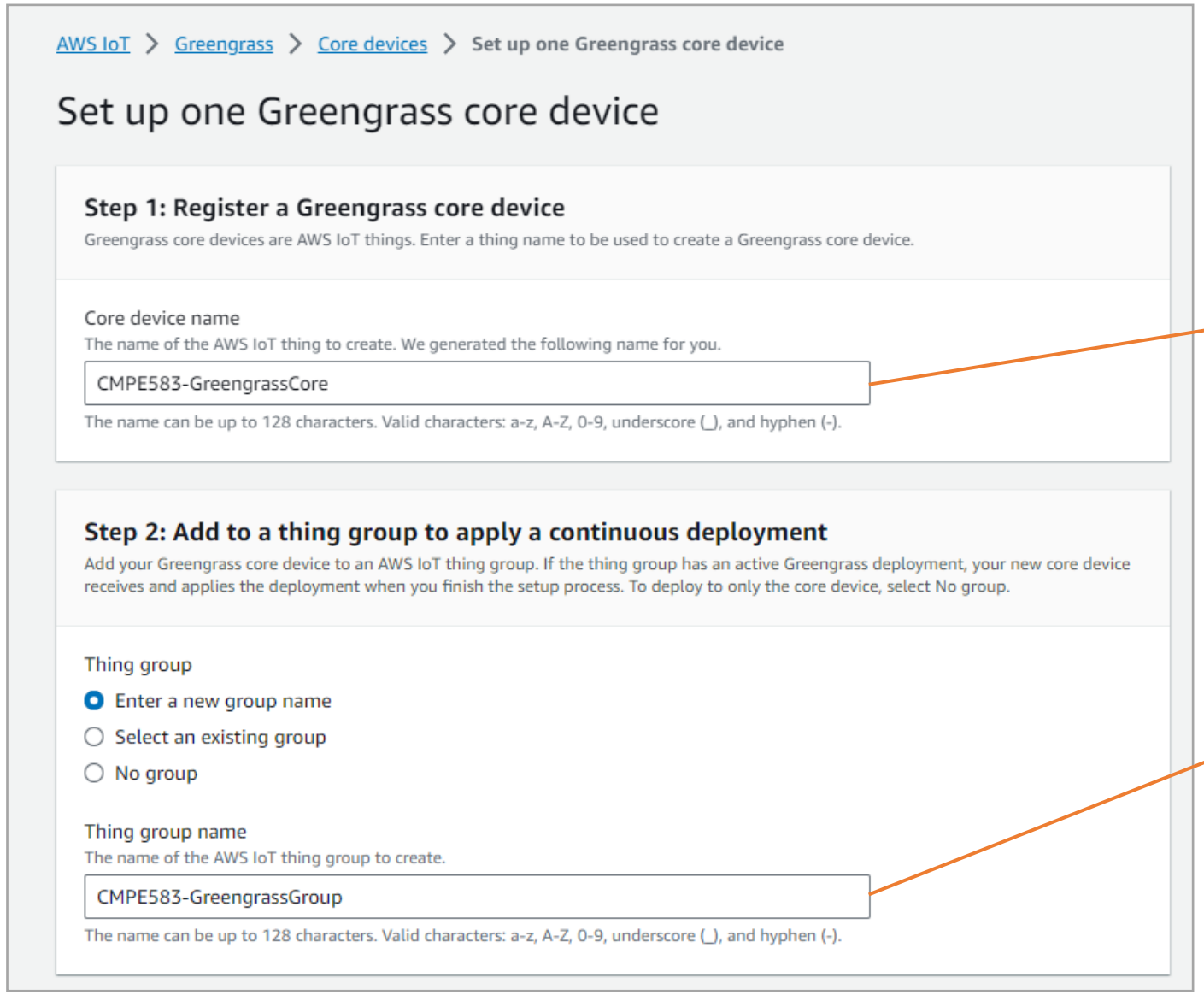
Copy the Access Keys



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 4: Install the Greengrass Core Software

Configure AWS Credentials on the Client

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

  Provide credentials as environment variables before running the installer:

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

- Credentials are not saved by the Greengrass installer!

Step 4: Install the Greengrass Core Software

For Linux Based Devices

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

Download the installer:

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

Run the installer:

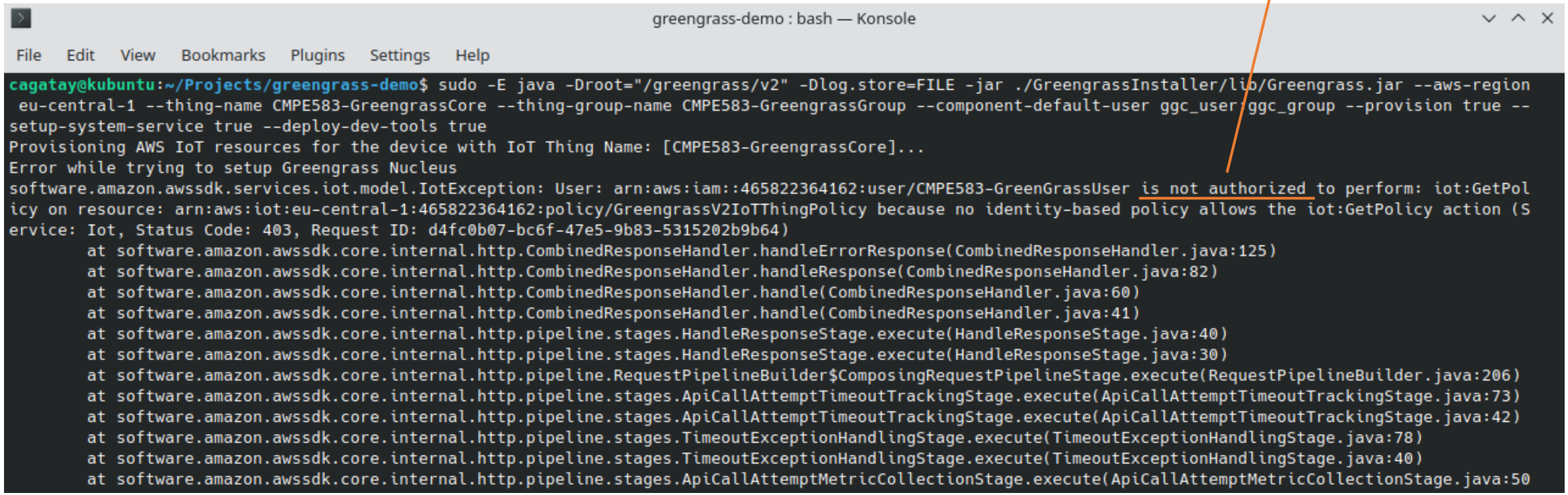
```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE -jar ./GreengrassInstaller/lib/Greengrass.jar --aws-
    region eu-central-1 --thing-name CMPE583-GreengrassCore --thing-group-name CMPE583-
    GreengrassGroup --component-default-user ggc_user:ggc_group --provision true --setup-system-service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

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

Step 4: Install the Greengrass Core Software

Run the Installer Command

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

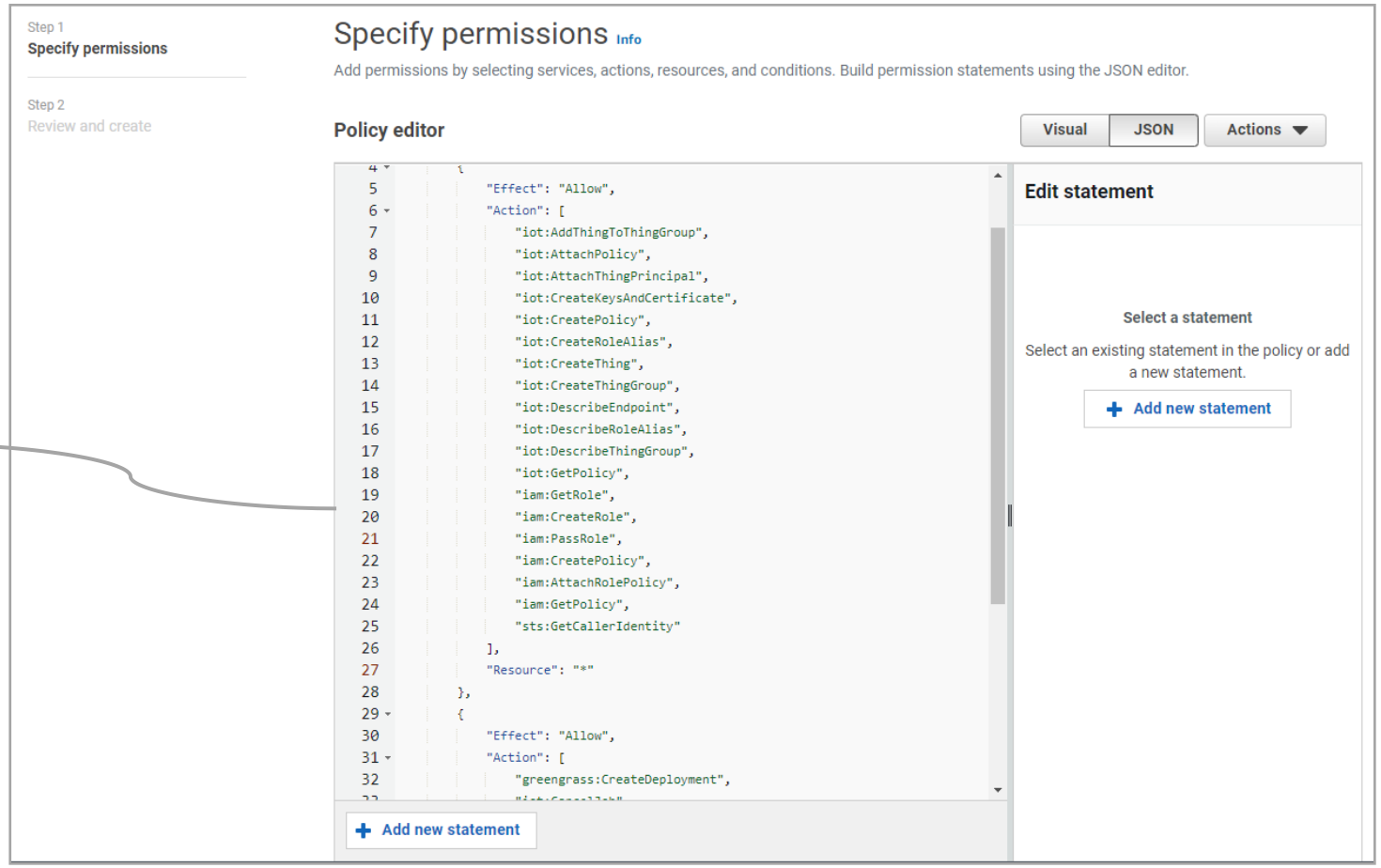


Step 4: Install the Greengrass Core Software

Set Required Permissions to Greengrass User

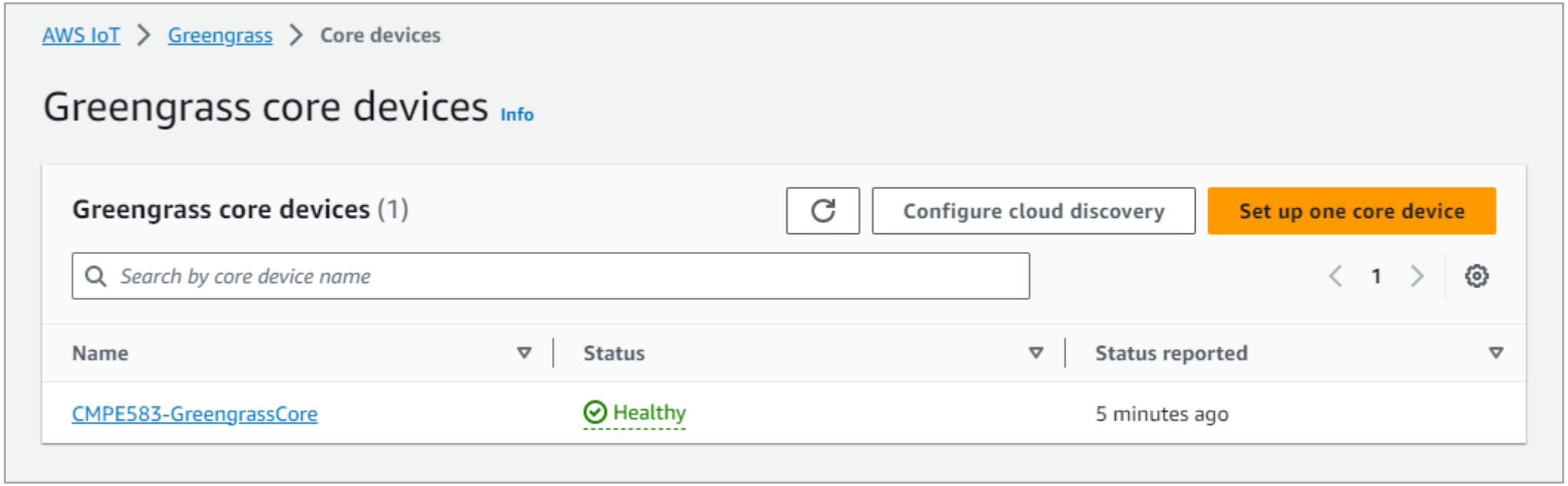
- When installing a core device, the user we created needs certain permissions to provision AWS IoT resources.

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

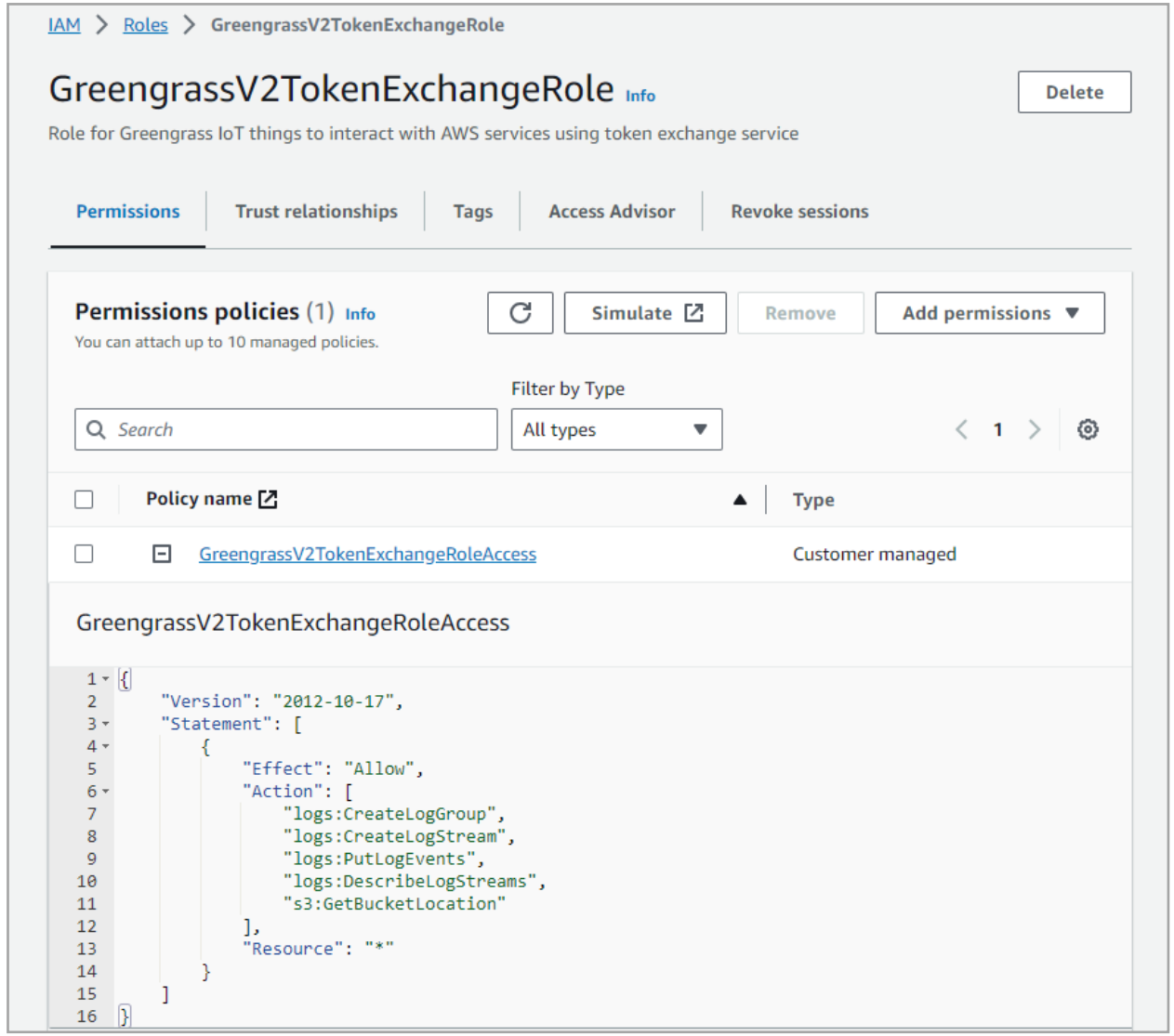


Step 4: Install the Greengrass Core Software

Rerun the Installer and Check If Core Device Is Healty



Step 4: Install the Greengrass Core Software



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

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

Step 4: Install the Greengrass Core software

Troubleshooting

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

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

Step 4: Install the Greengrass Core software

Monitor AWS IoT Greengrass logs

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

```
/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

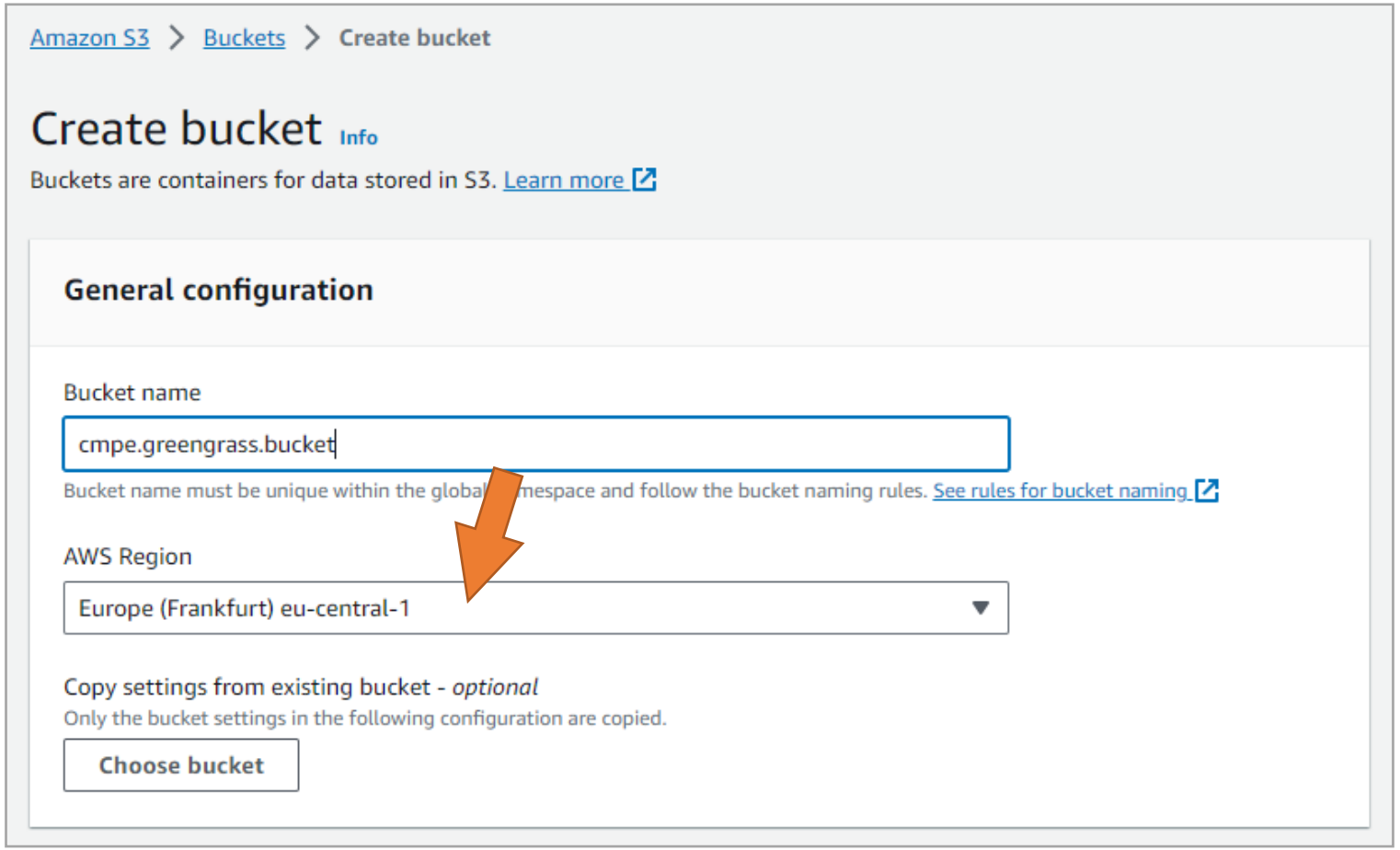
- Every component contains a recipe file.
- Recipes can be defined in JSON or YAML format.
- It defines component's metadata.
  - Configuration parameters
  - Component dependencies
  - Lifecycle
  - Platform compatibility

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

Step 1: Upload the Artifacts

Create a bucket via AWS S3 (Console)

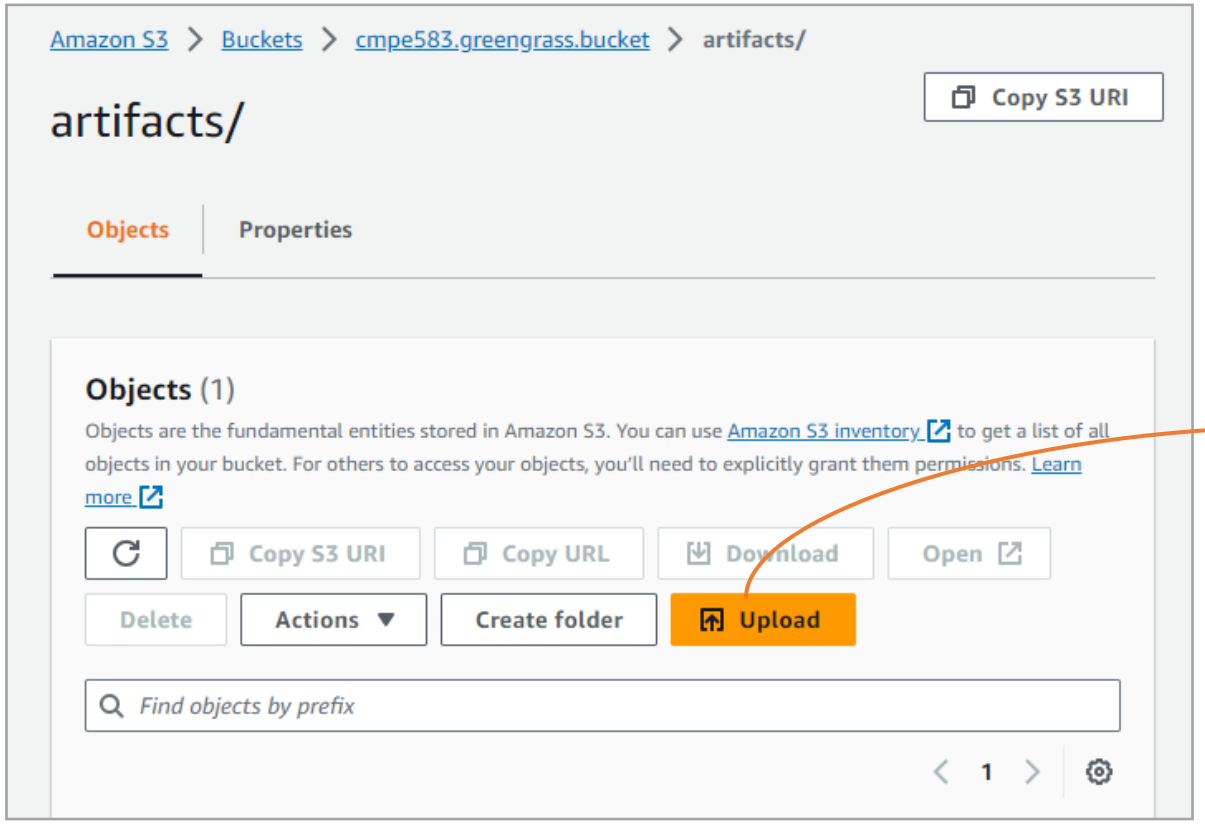


- Be sure that the S3 bucket is in the same AWS Region where you create the component.
- AWS IoT Greengrass doesn't support cross-Region requests for component artifacts!

Step 2: Upload the Artifacts

Create artifacts folder & upload files

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



cmpe583-PrintMsg.py

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

Step 2: Set Policies to Access Files in S3

Via AWS IAM (Console)

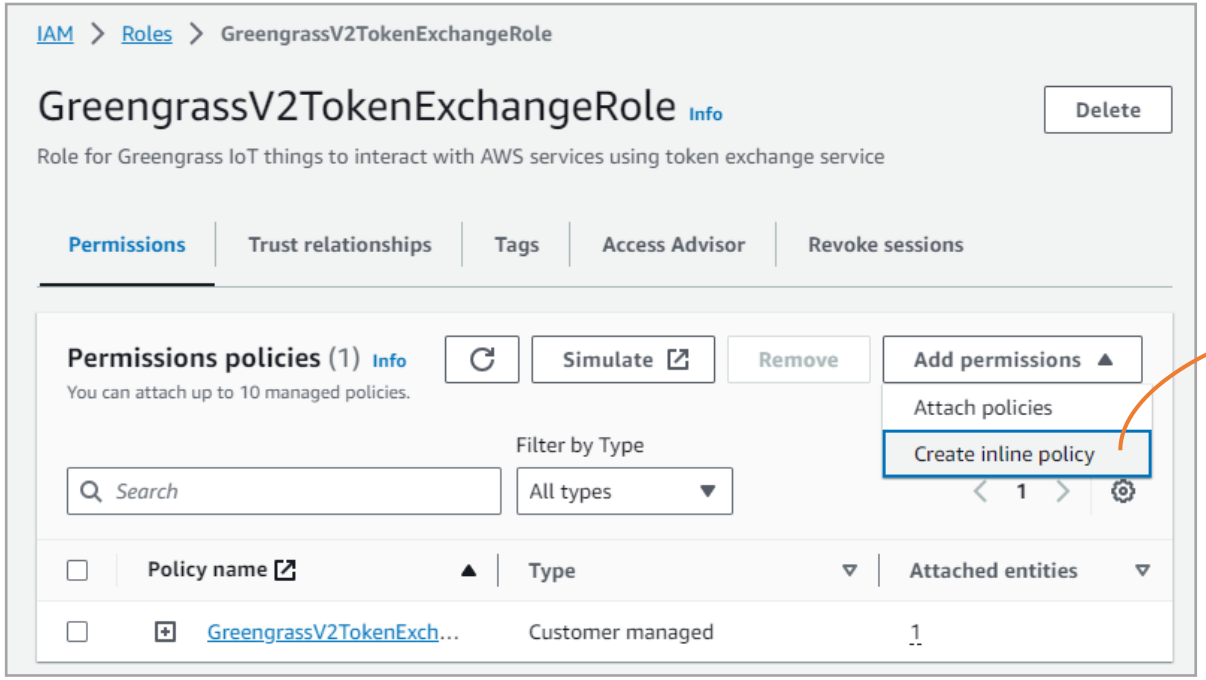
- Greengrass component artifacts should be stored in AWS S3 bucket.
- Therefore, you should allow the core device to access component artifacts in the S3 bucket.
  - Attach a policy similar to policy below to GreengrassV2TokenExchangeRole from AWS IAM console:

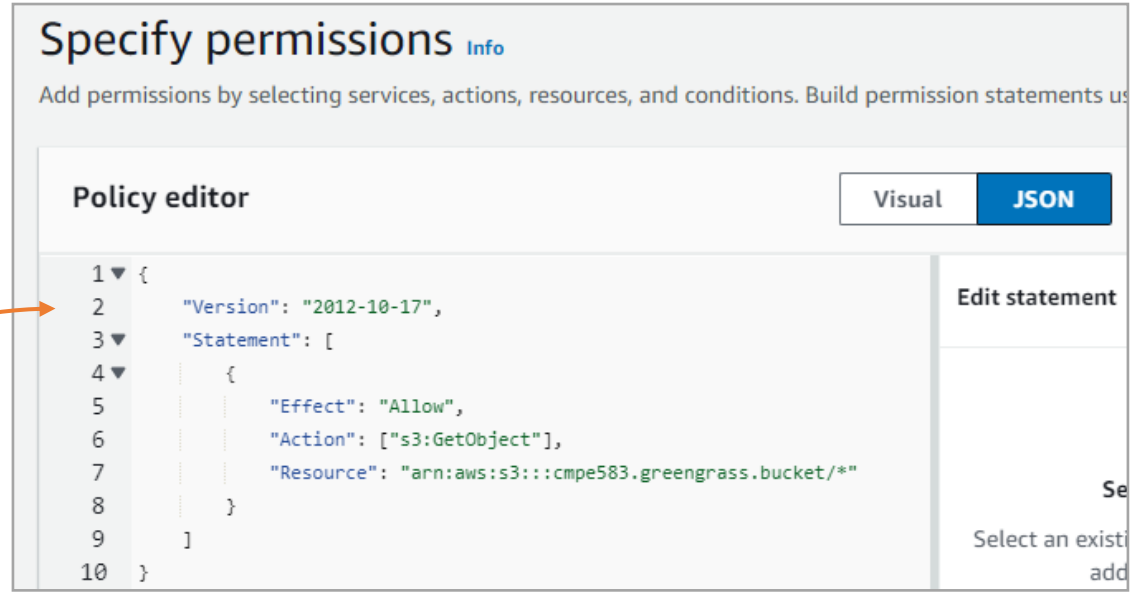
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "s3:GetObject" ],
      "Resource": "arn:aws:s3:::Bucket-Name/*"
    }
  ]
}
```

Step 2: Set Policies to Access Files in S3

Via AWS IAM (Console)

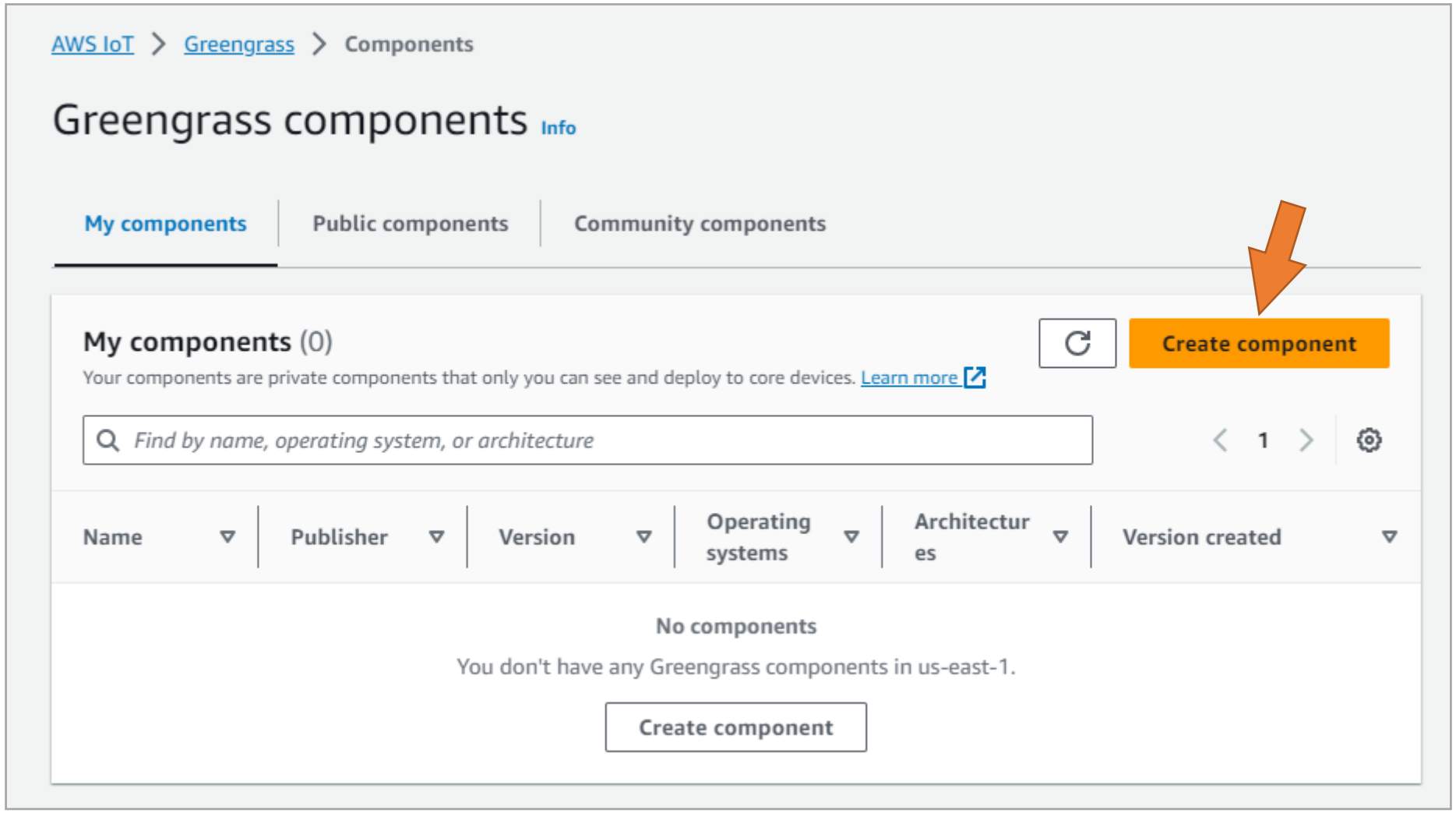
- You can create a new policy or simply attach an inline policy to related role in order to allow core device to Access objects in the S3 bucket.





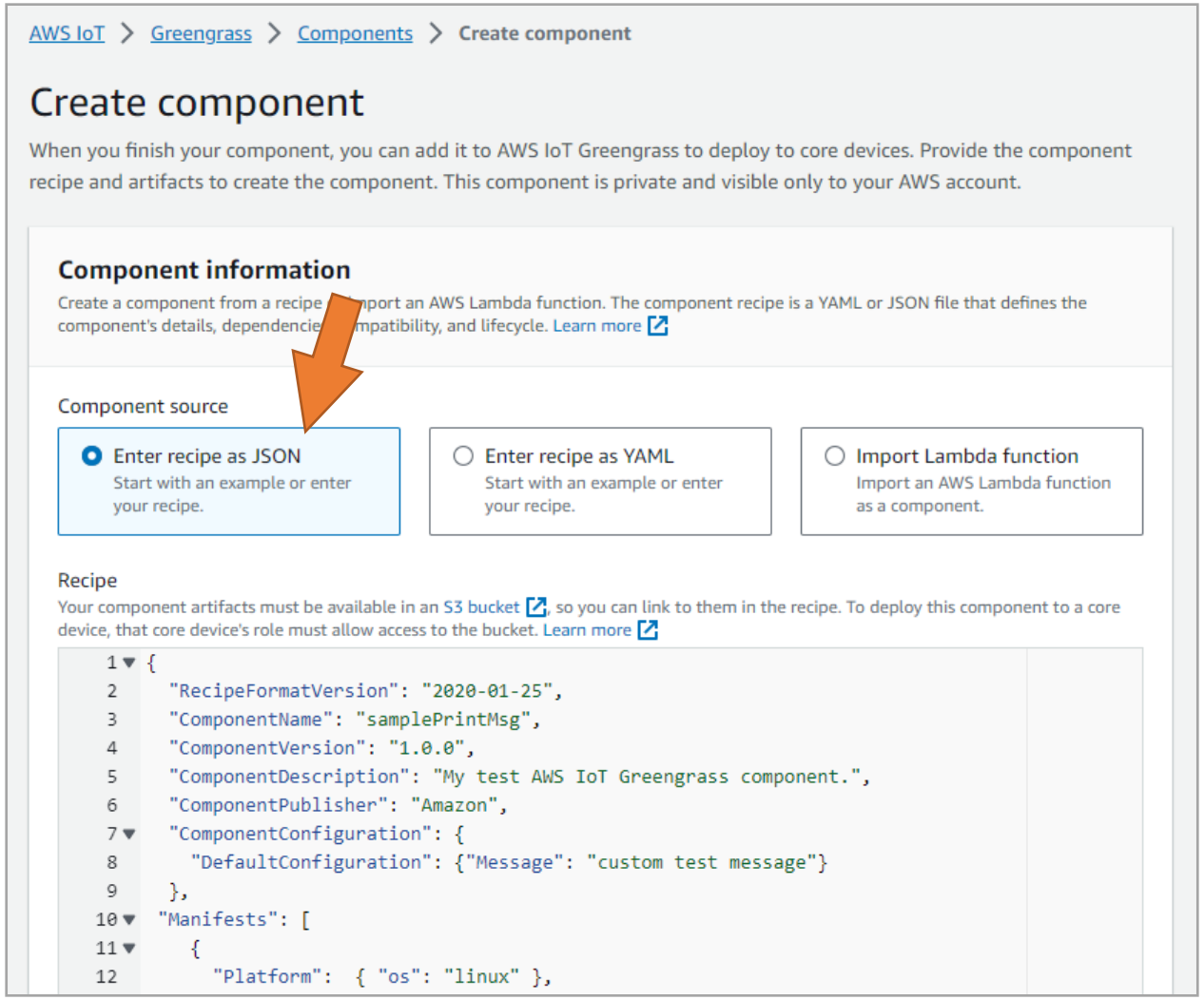
Step 3: Create the Component

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

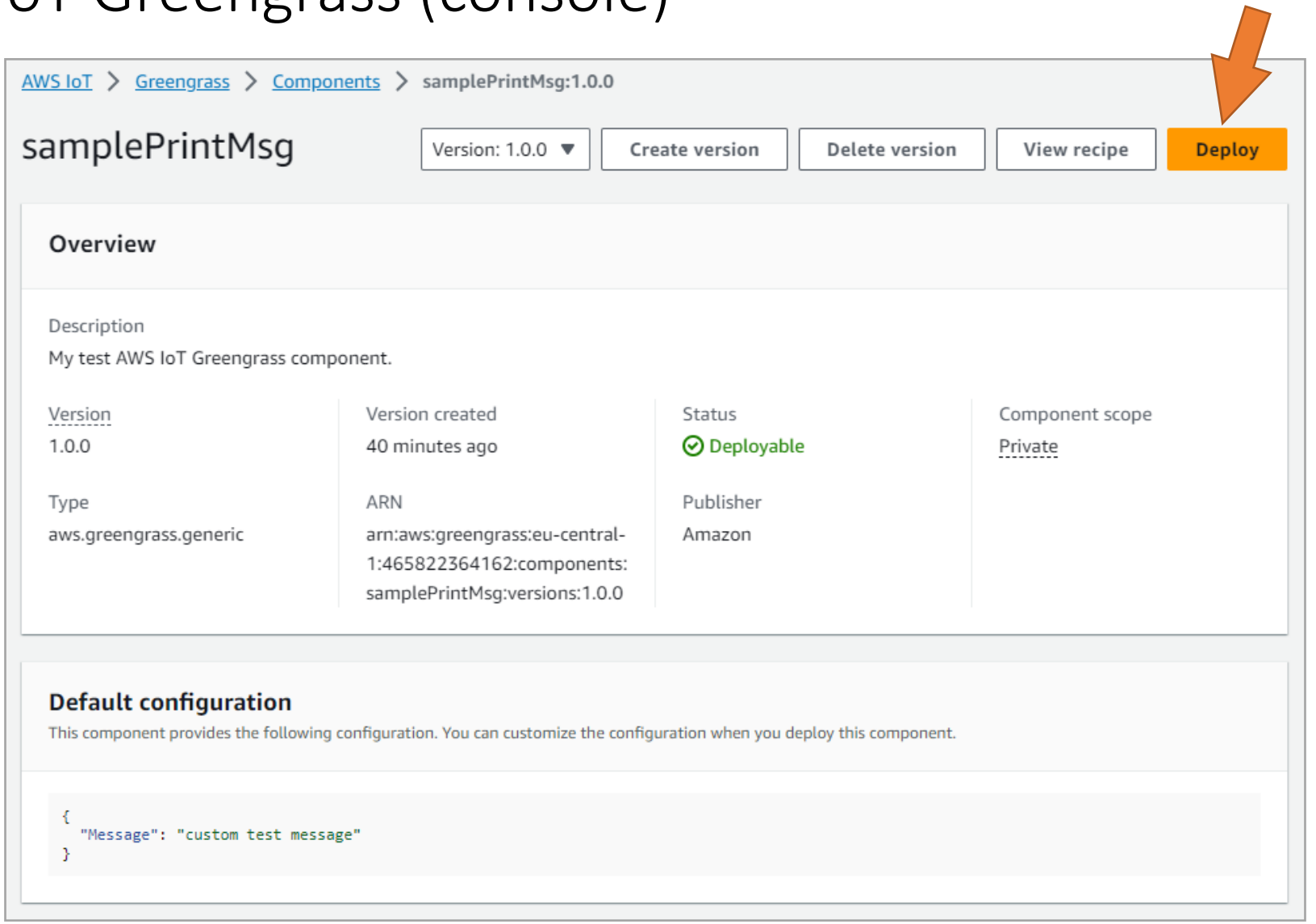
This test will be run on Linux device. So other OSs are ignored!

Define how to execute your artifact

Provide the artifact URI

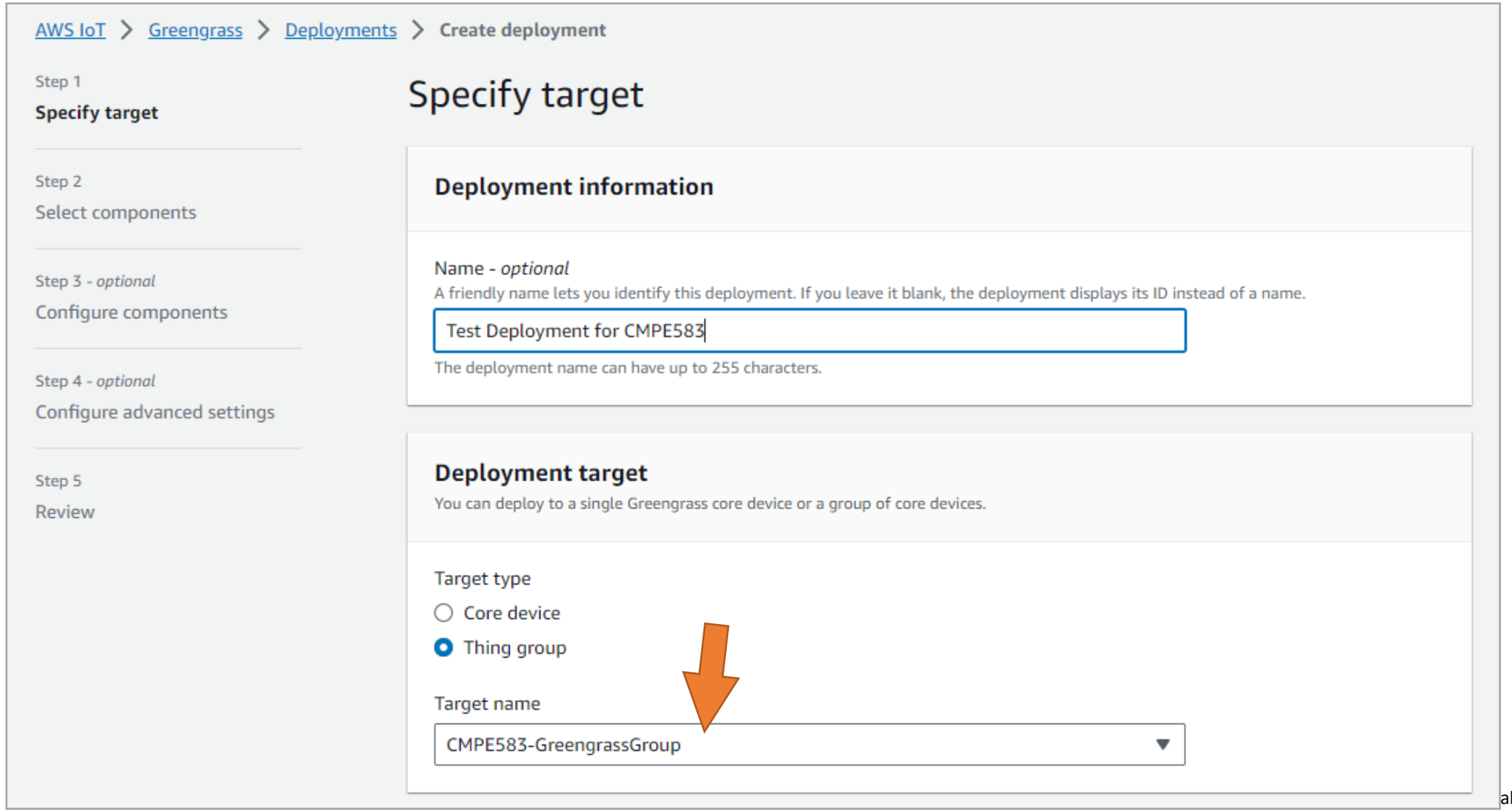
Step 4: Deploy the Component

Via AWS IoT Greengrass (console)



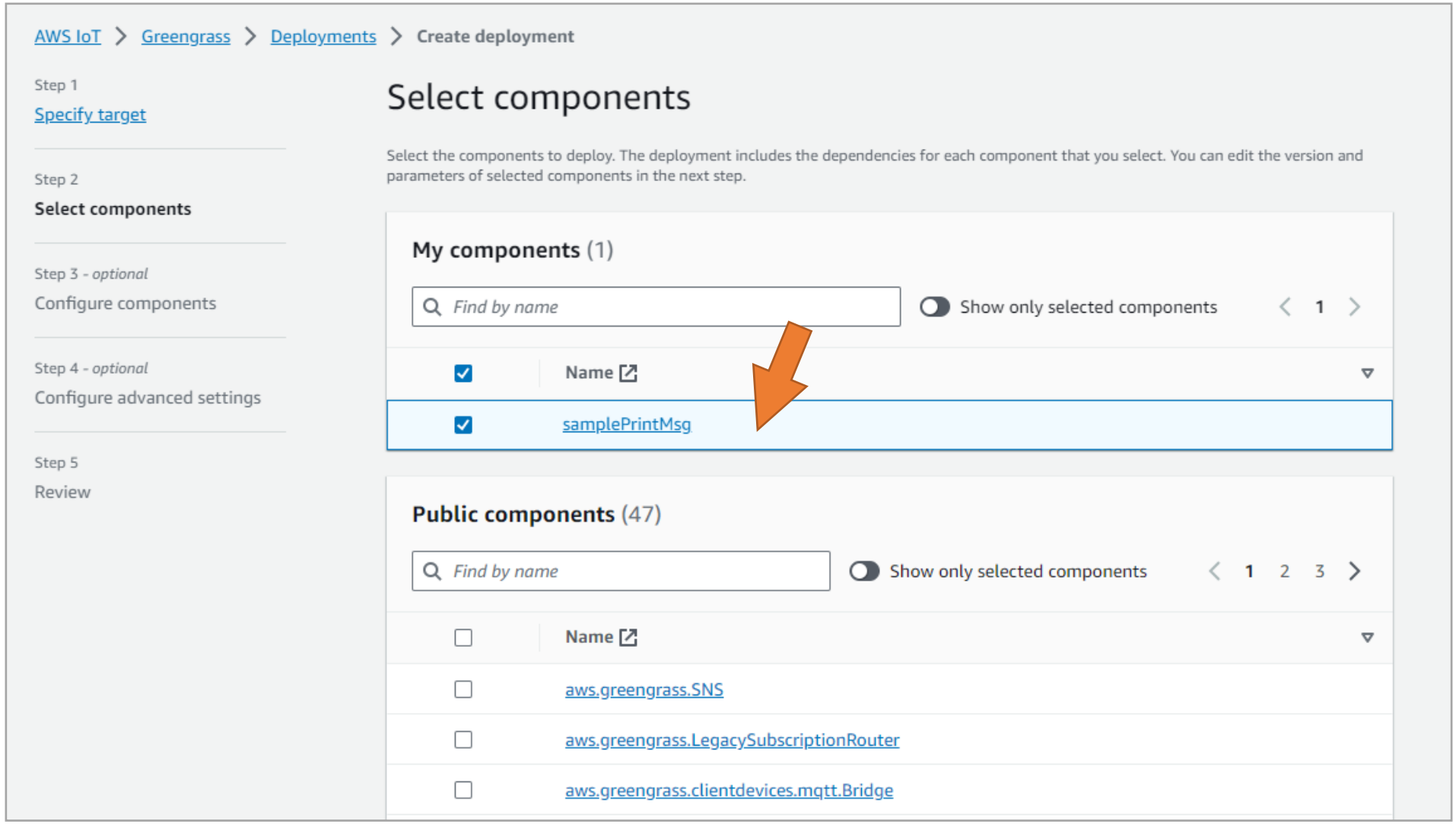
Step 4: Deploy the Component

Configure Deployment



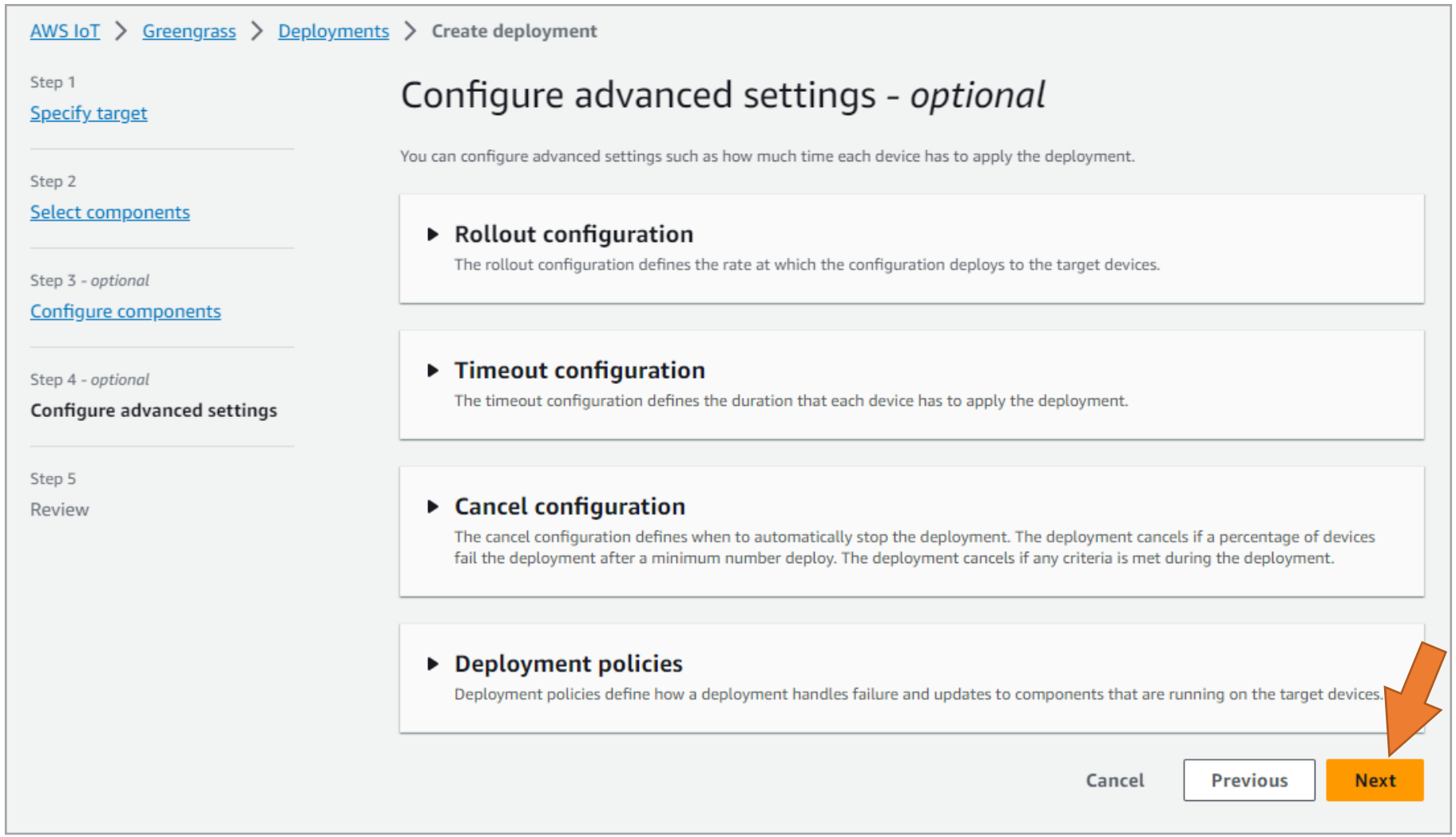
Step 4: Deploy the Component

Select the Component



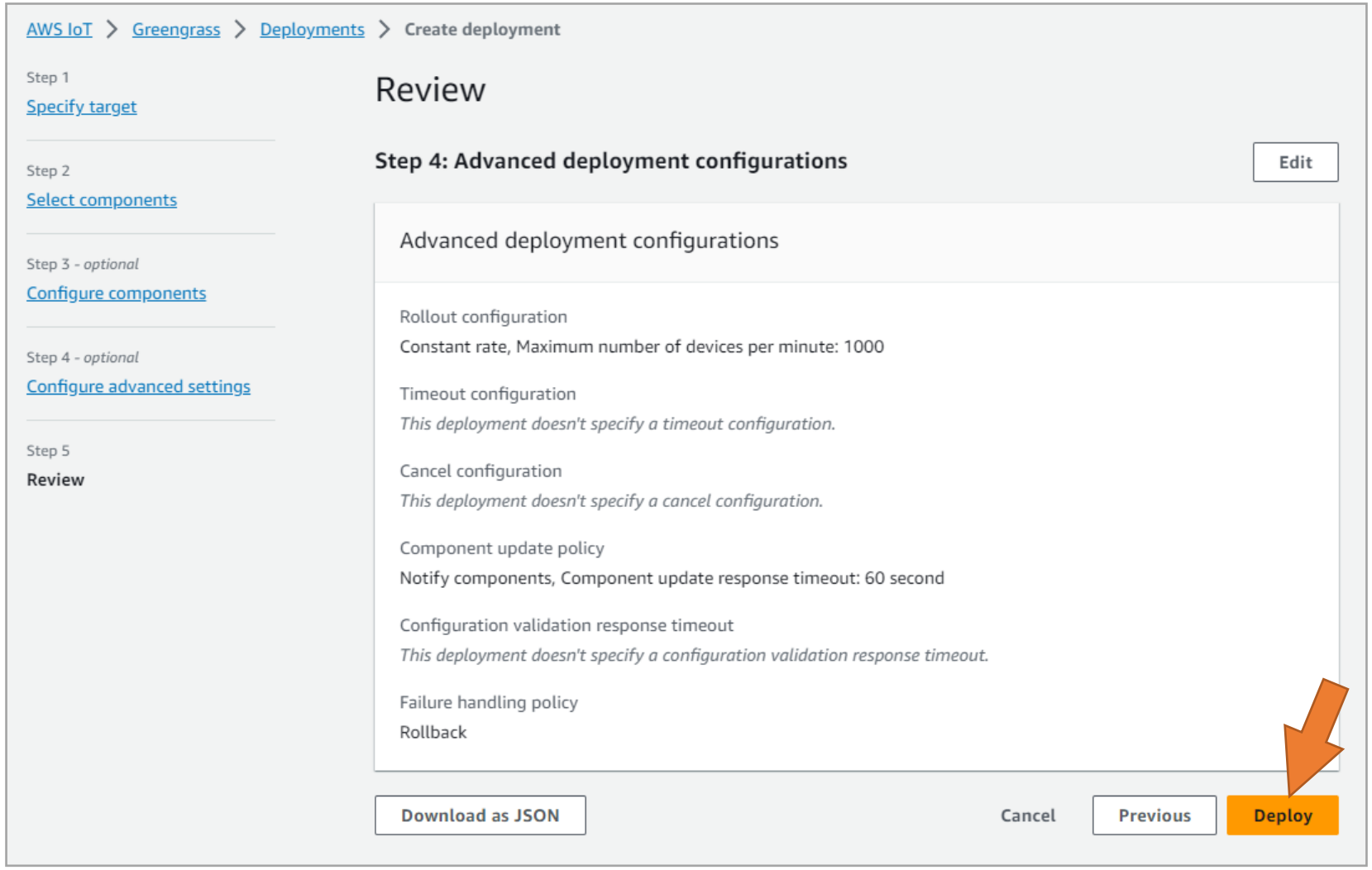
Step 4: Deploy the Component

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

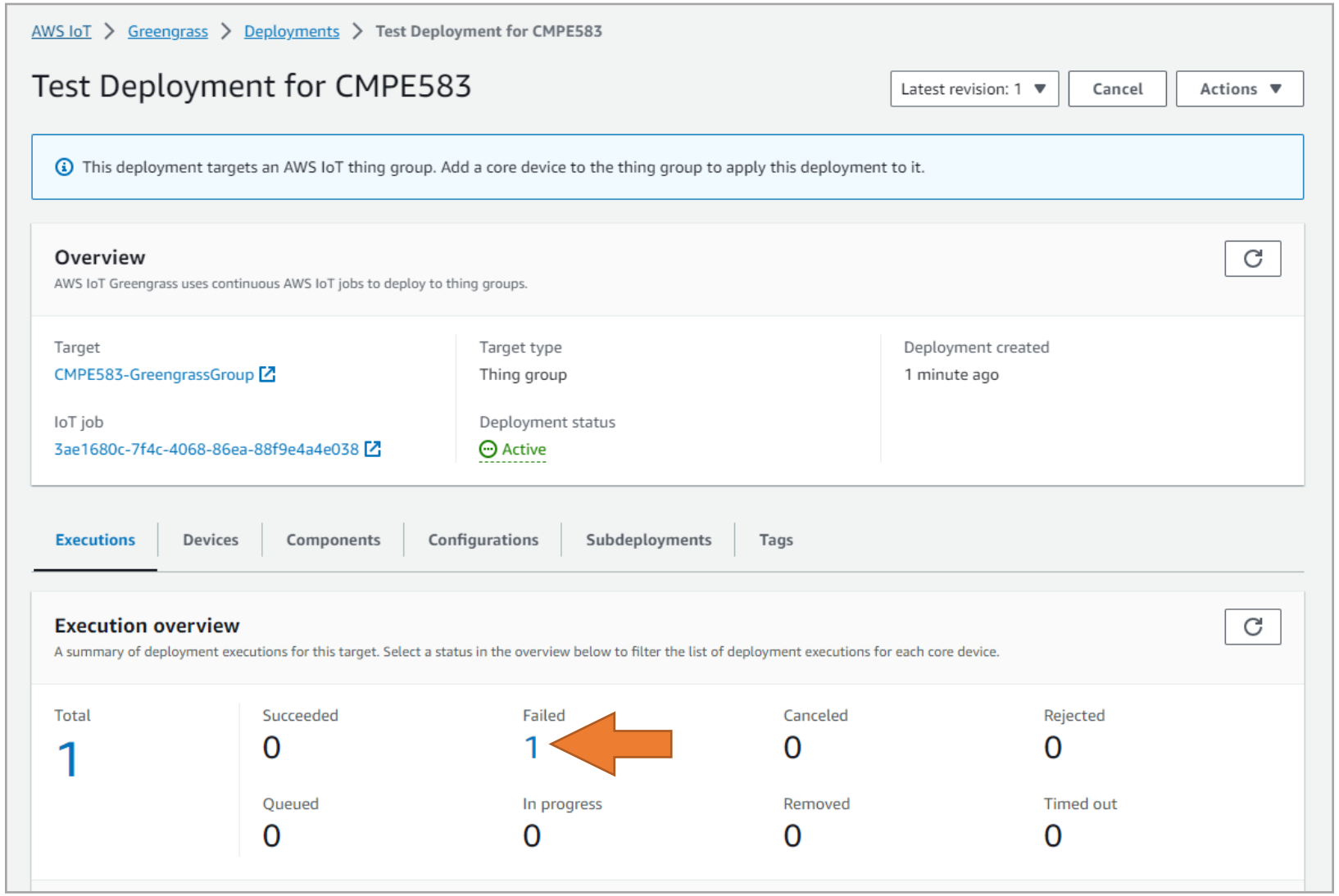
Check Log Files

- Check the output log file created by your compenent to verify everthing 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!

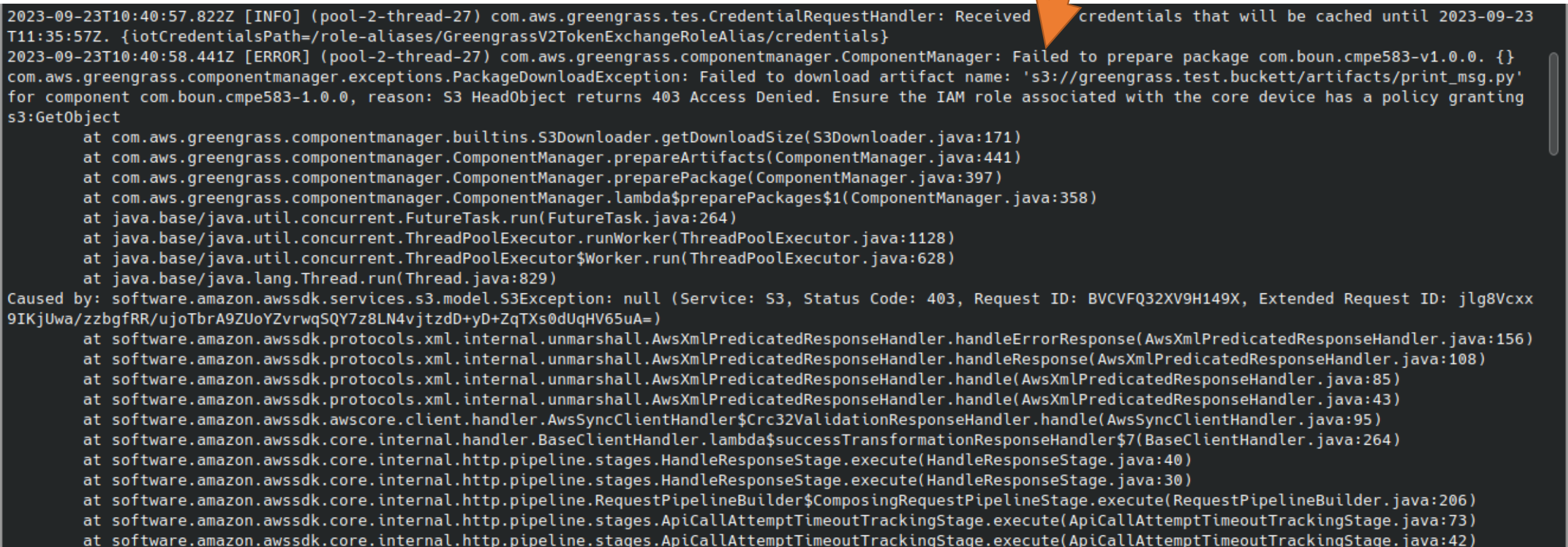
Troubleshooting



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  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.

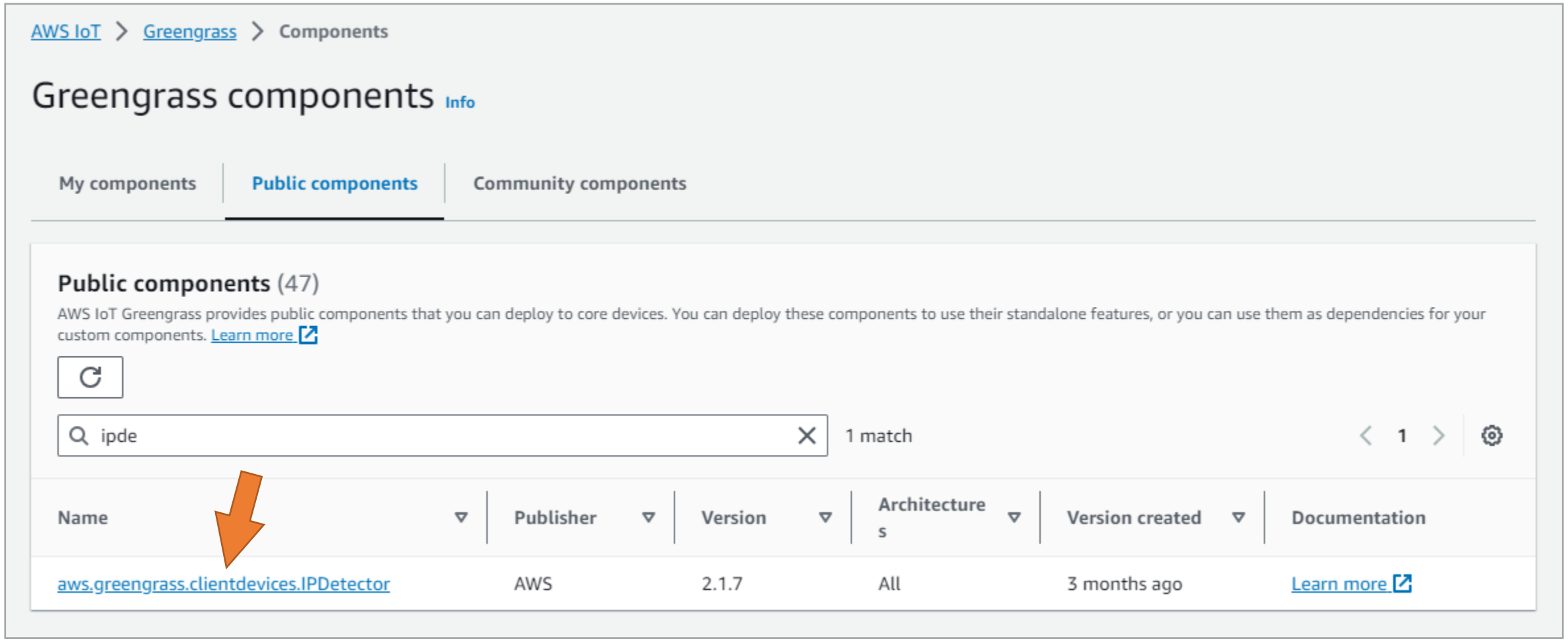
AWS Provided Components

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

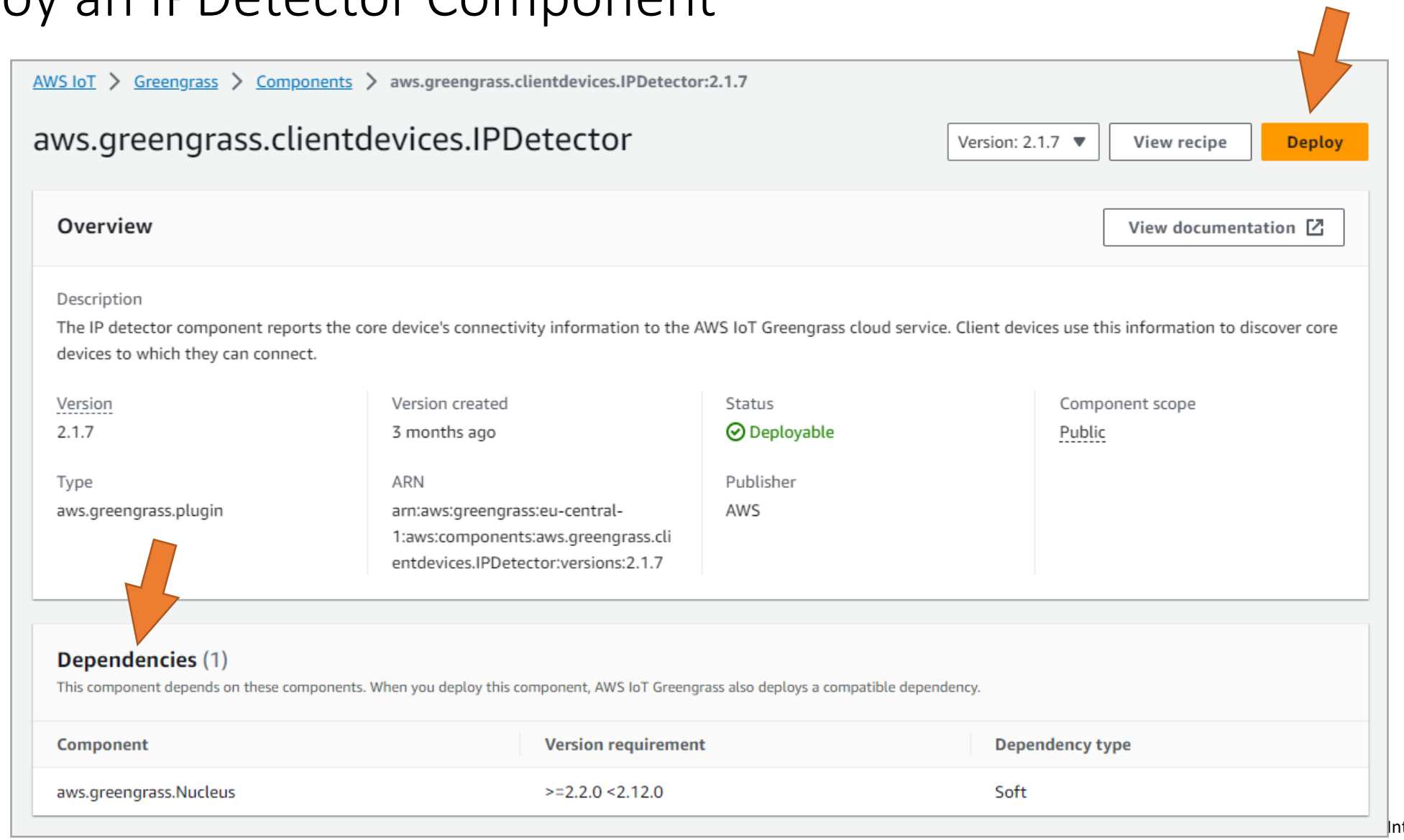
Collecting IP Addresses of Core Devices

IPDetector



Collecting IP Addresses of Core Devices

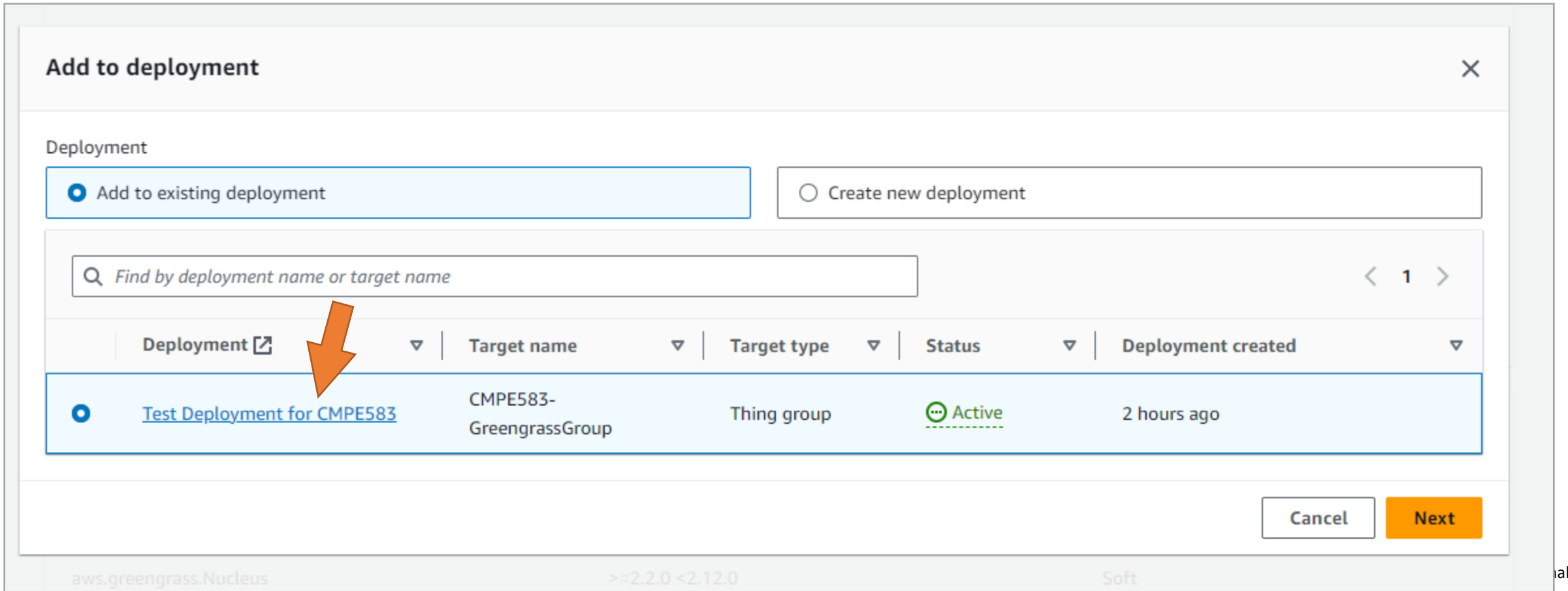
Deploy an IPDetector Component



Collecting IP Addresses of Core Devices

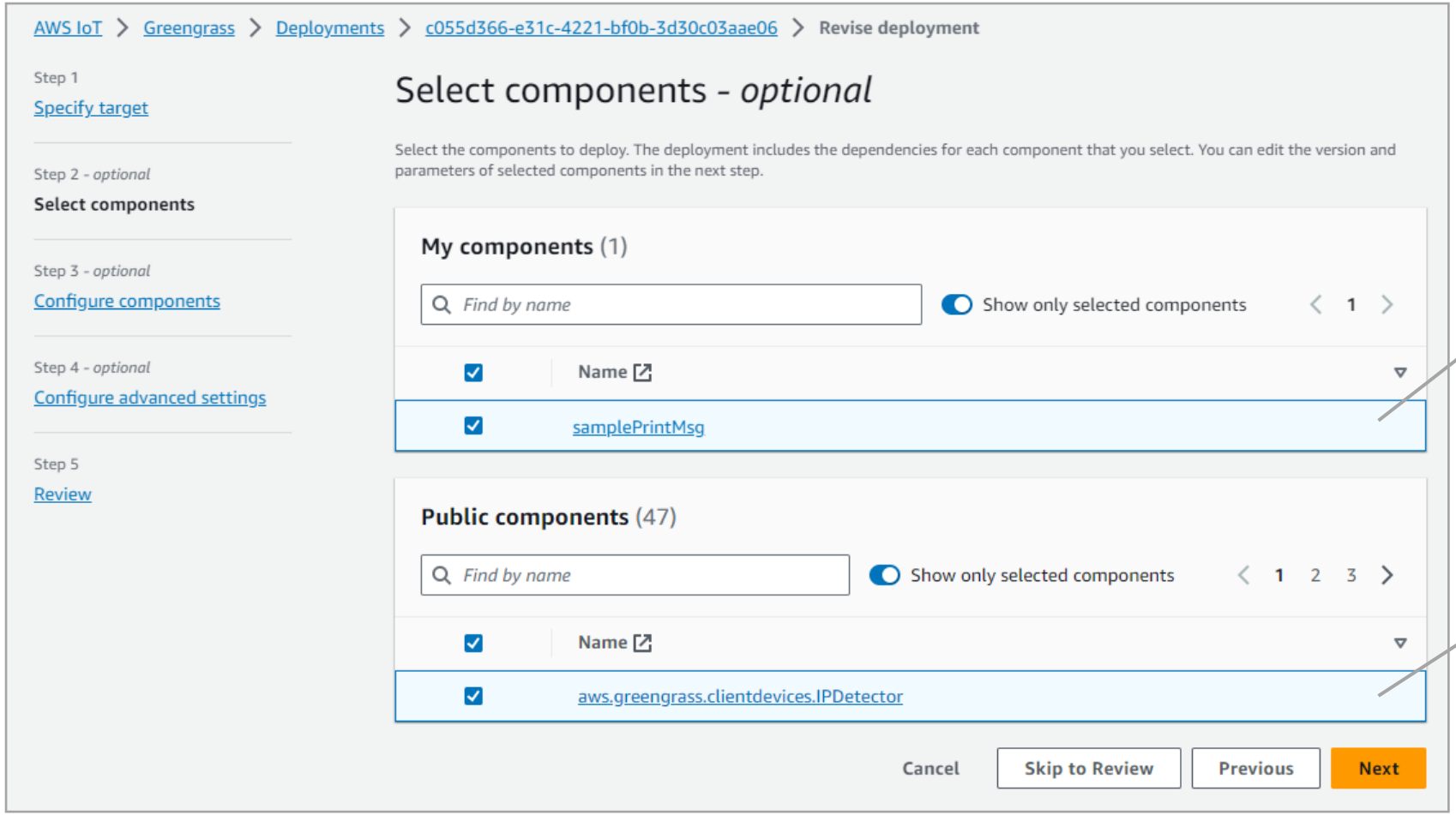
Add Public Component to Existing Deployment

- 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



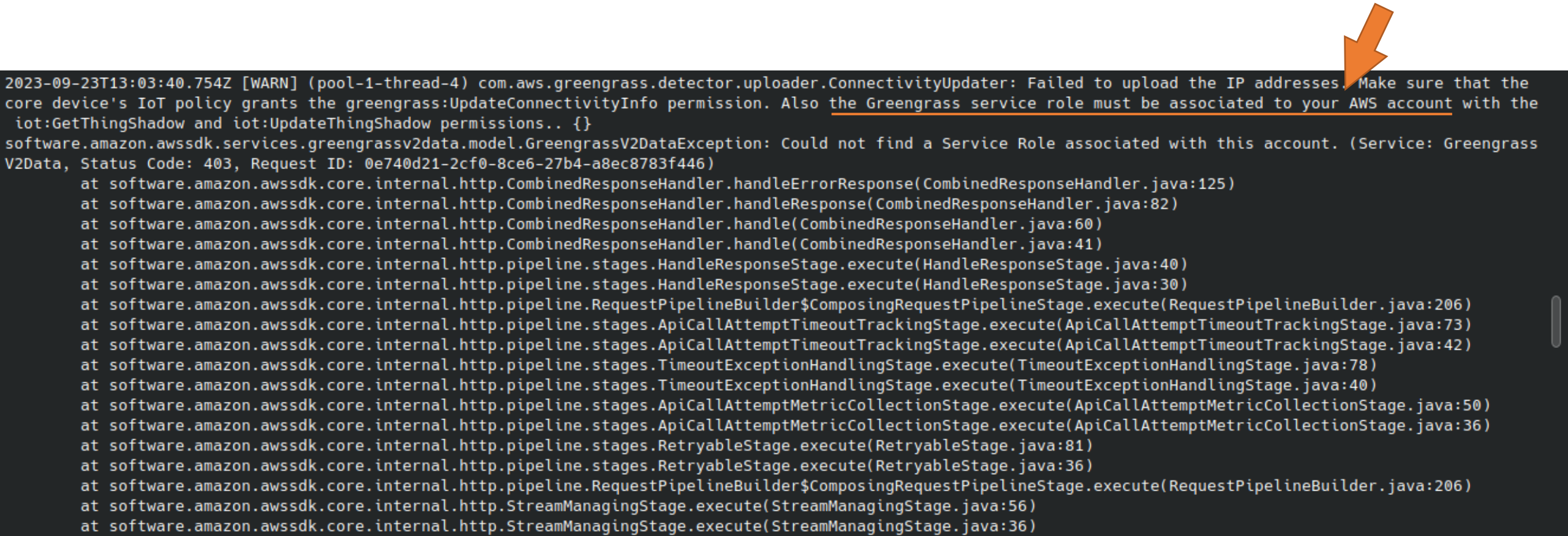
Custom component

AWS Provided component

Collecting IP Addresses of Core Devices

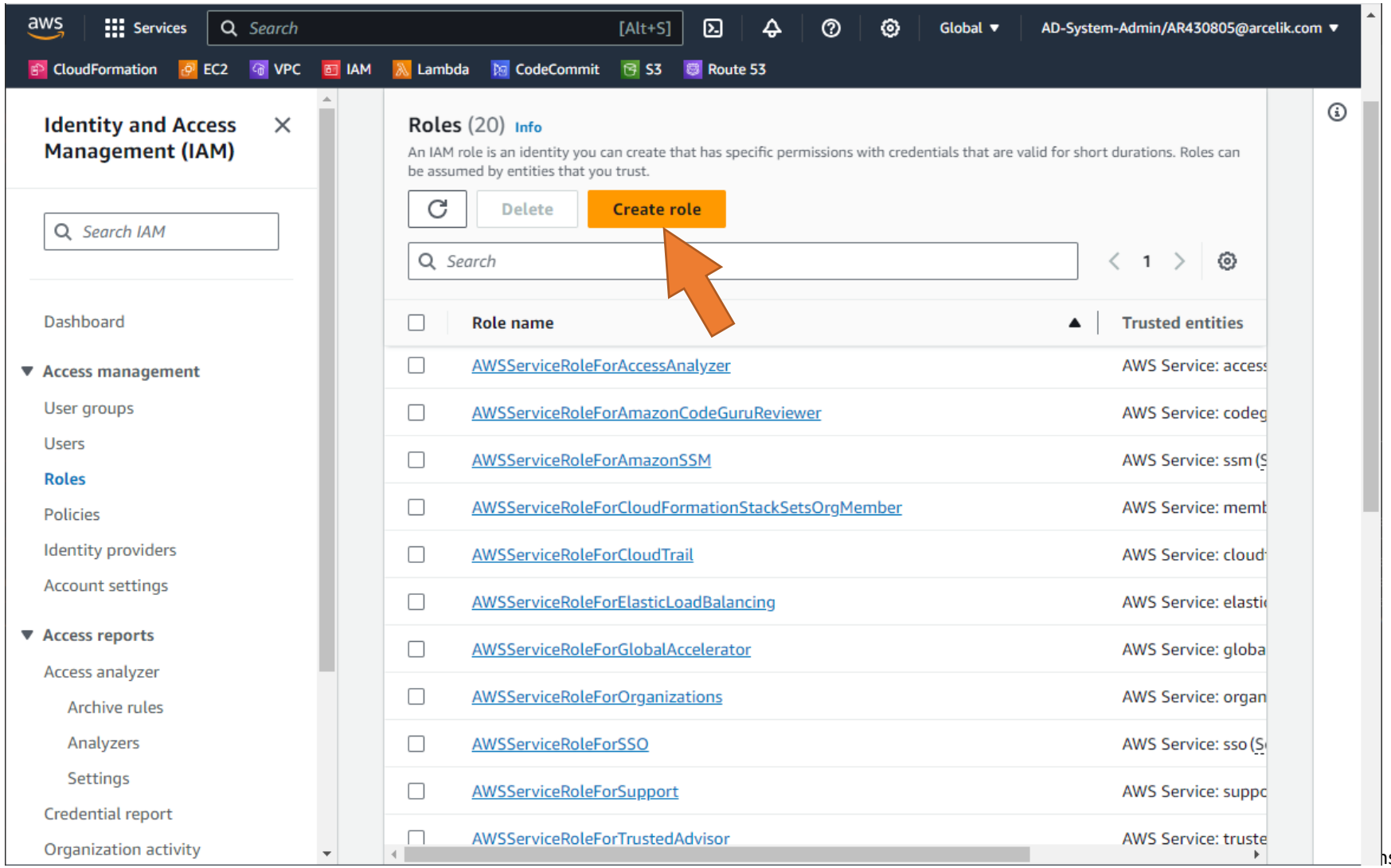
Troubleshooting

- 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.



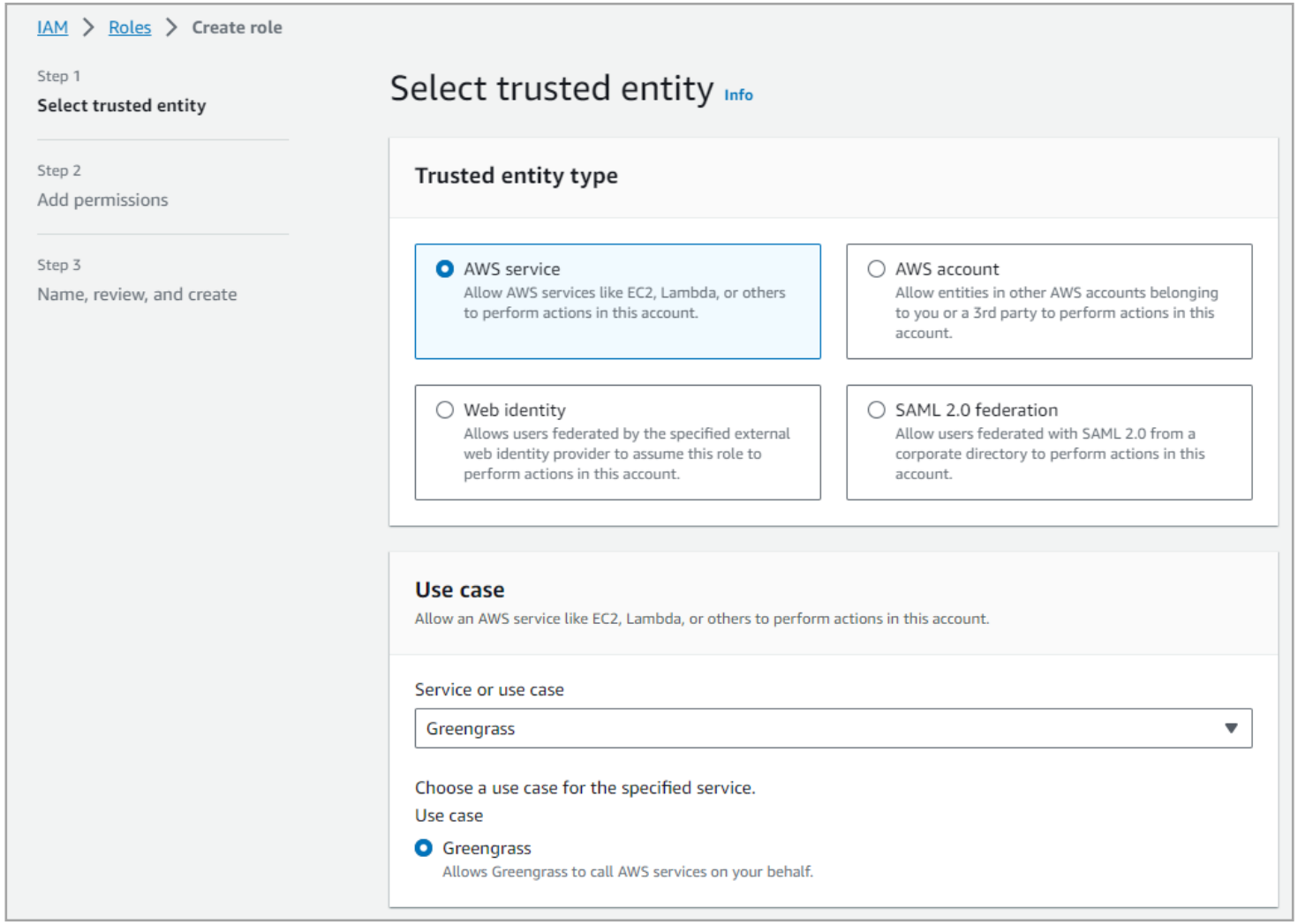
Troubleshooting

Create an IAM Role



Troubleshooting

Create an IAM Role

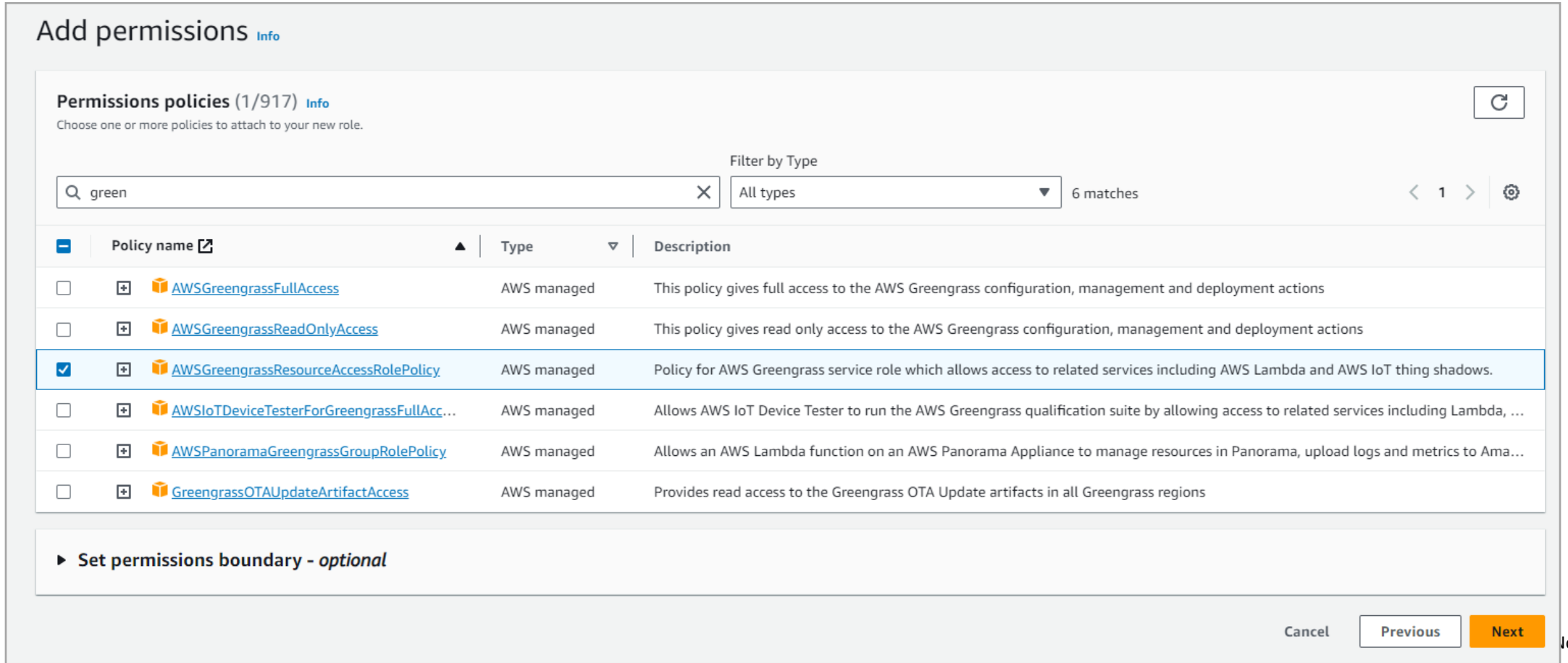


- Create a role that allows an AWS service to perform actions on your behalf.
- Allows Greengrass to call AWS services on your behalf.

Troubleshooting

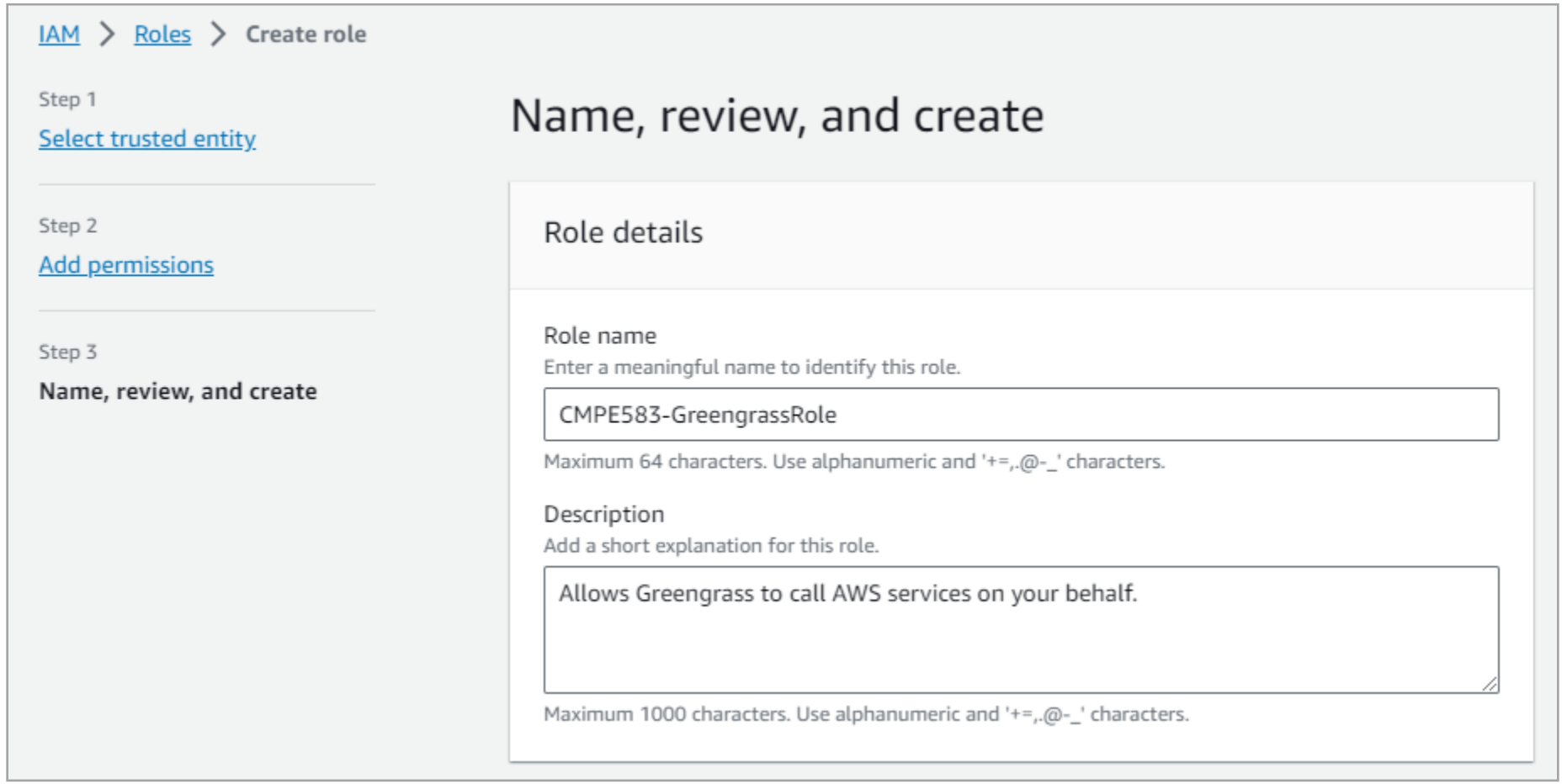
Add Permission to Created Role

- 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.



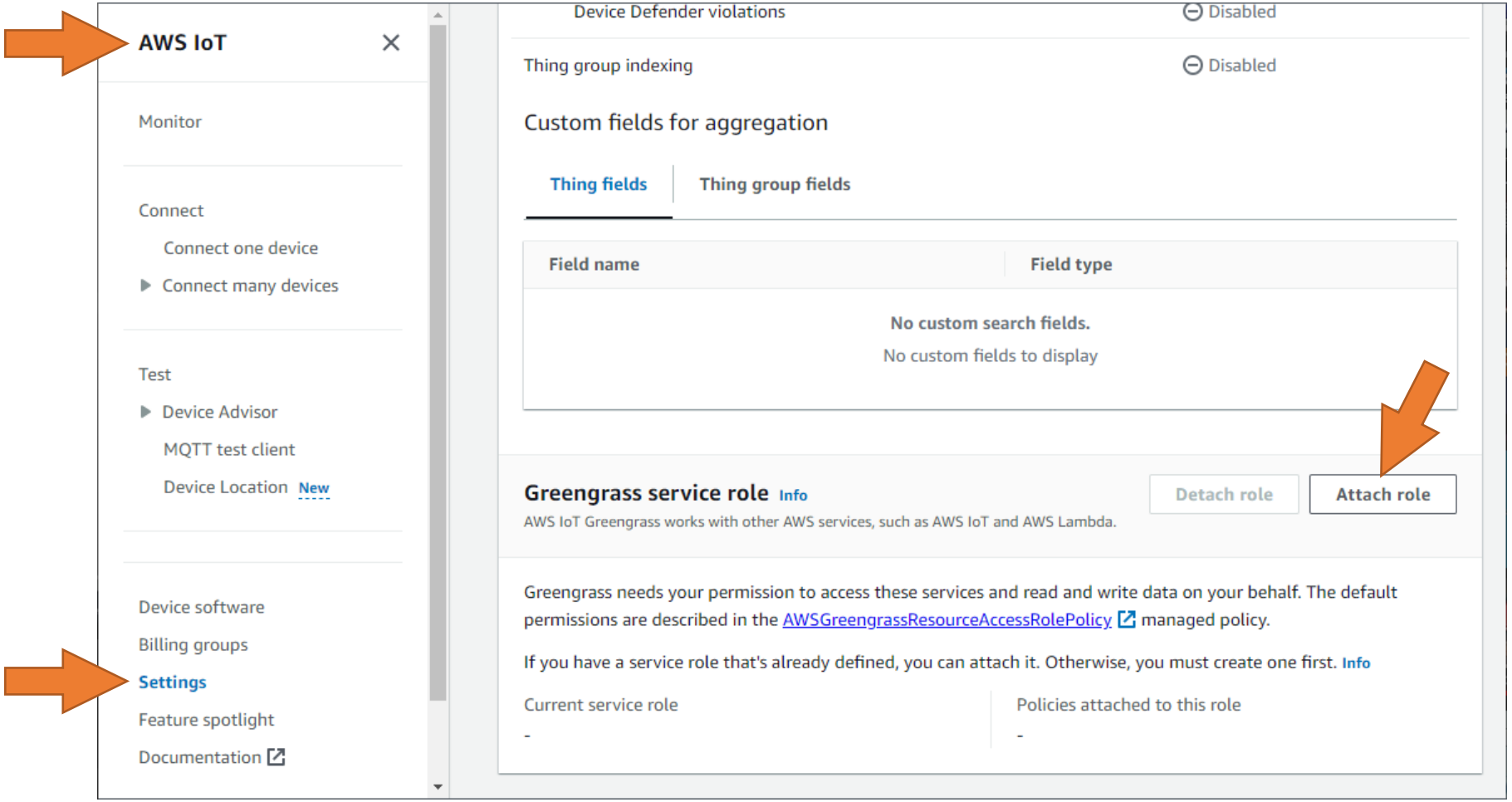
Troubleshooting

Review and Create Role



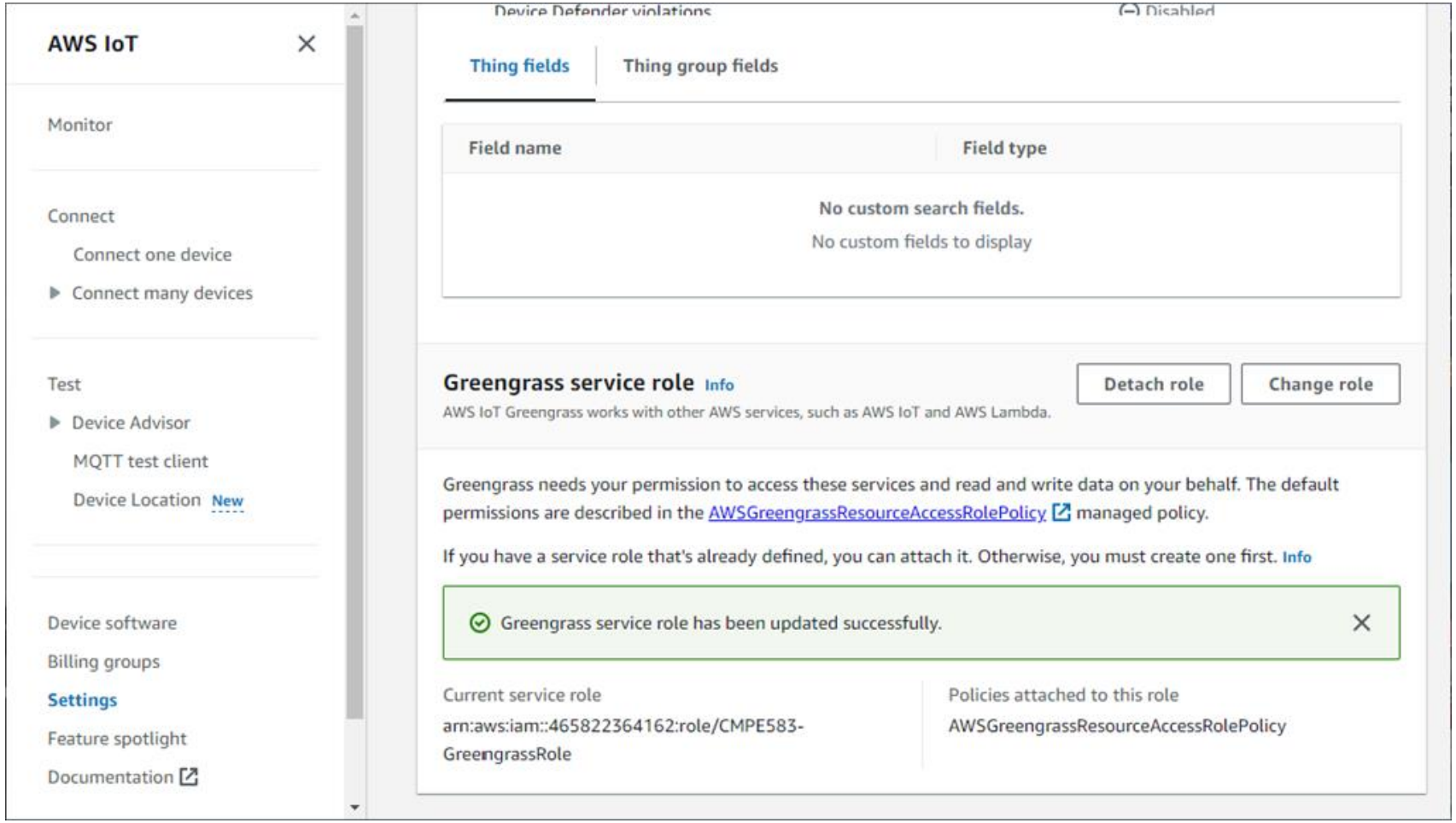
Troubleshooting

Attach Role to AWS IoT Service



Troubleshooting

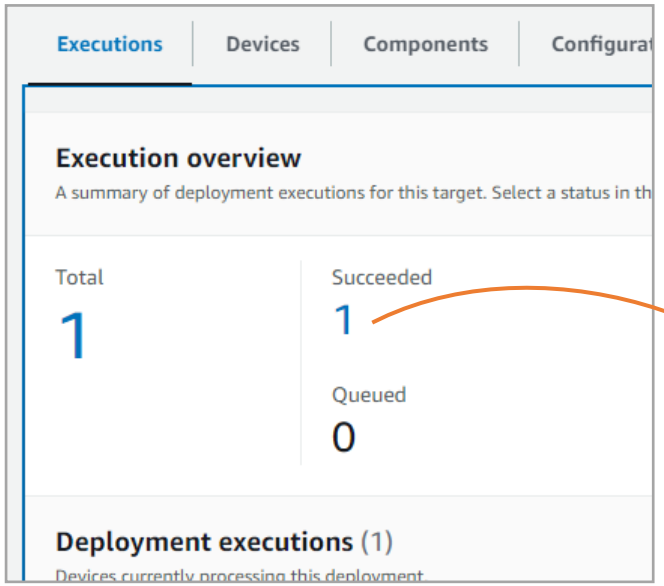
Attach Role to AWS IoT Service

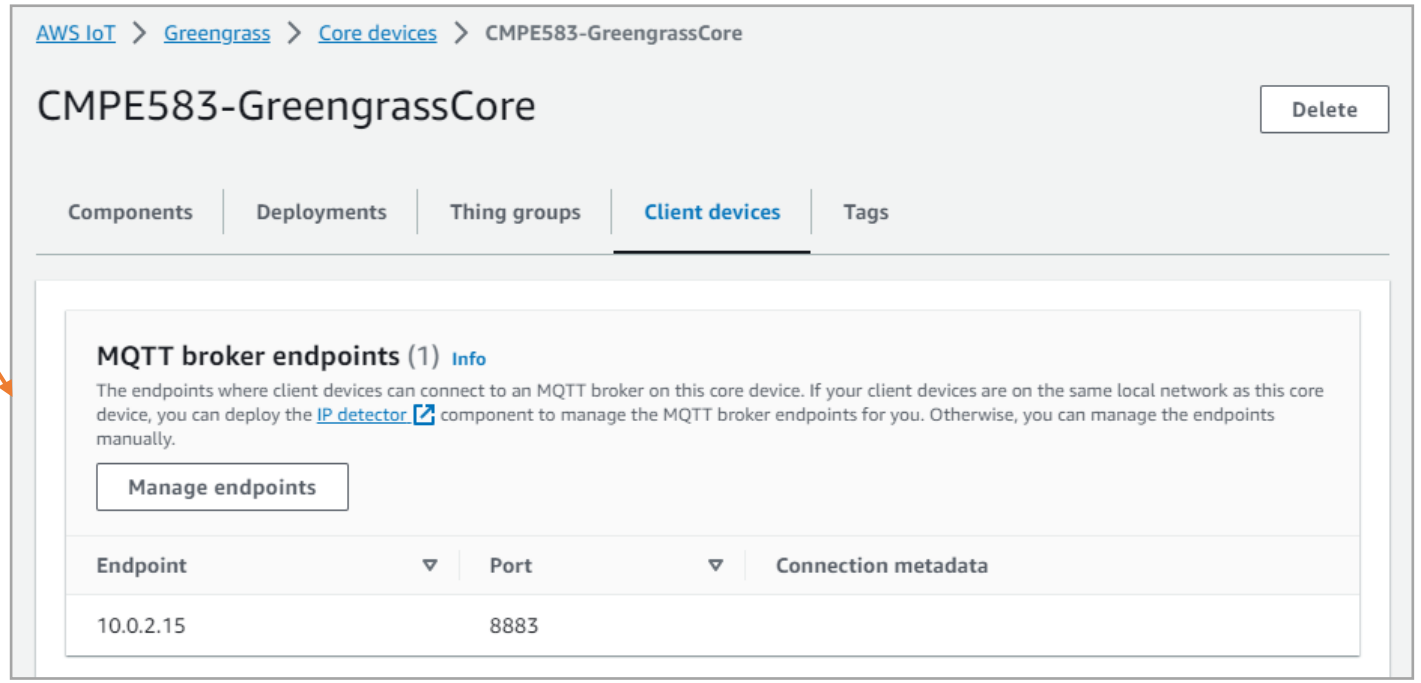


Troubleshooting

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.
- The IPC interface supports two types of operations:
  - **Request/response**
    - Components send a request to the IPC service and receive a response that contains the result of the request.
  - **Subscription**
    - Components send a subscription request to the IPC service and expect a stream of event messages in response.

AWS IoT Core MQTT Messaging IPC

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

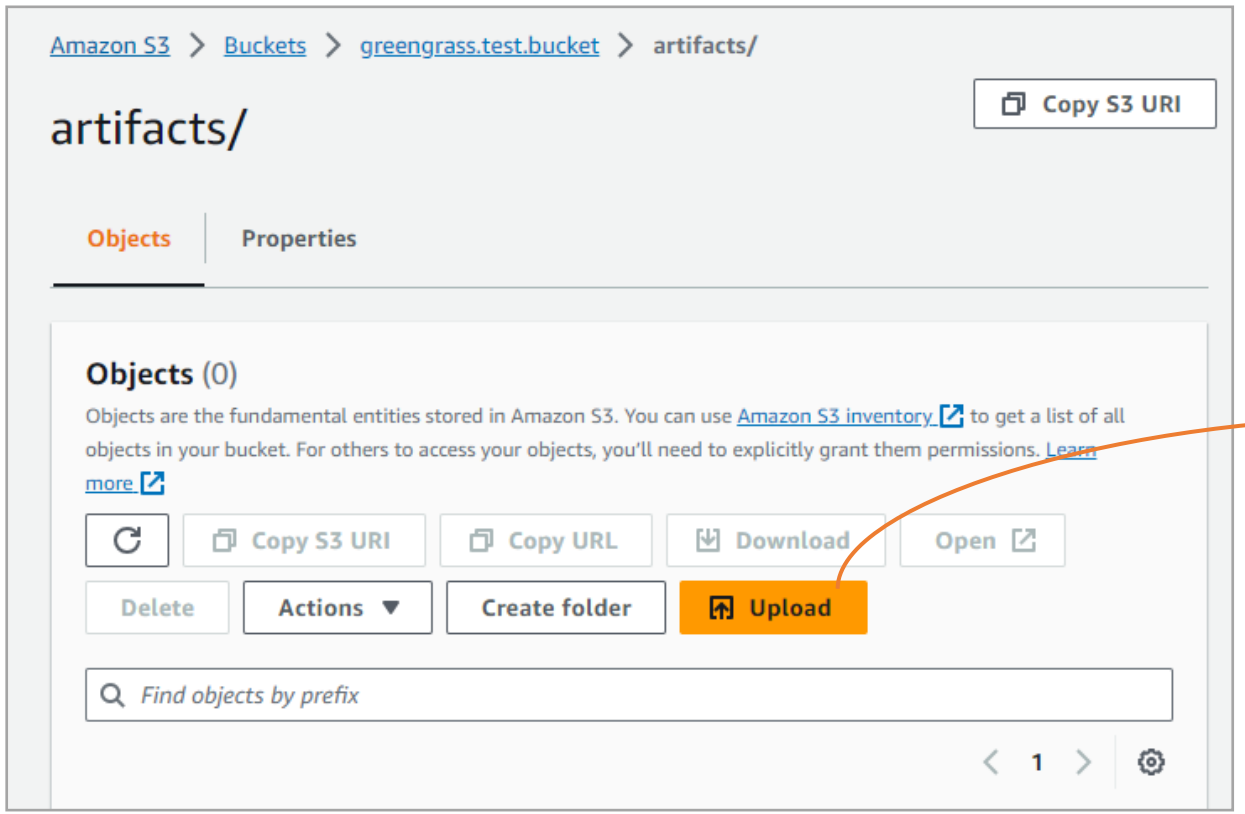
MQTT

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

Step 1: Develop & Upload Client-Side Code

Via AWS IoT Greengrass (Console)

- Upload your artifacts to a folder in your S3 bucket

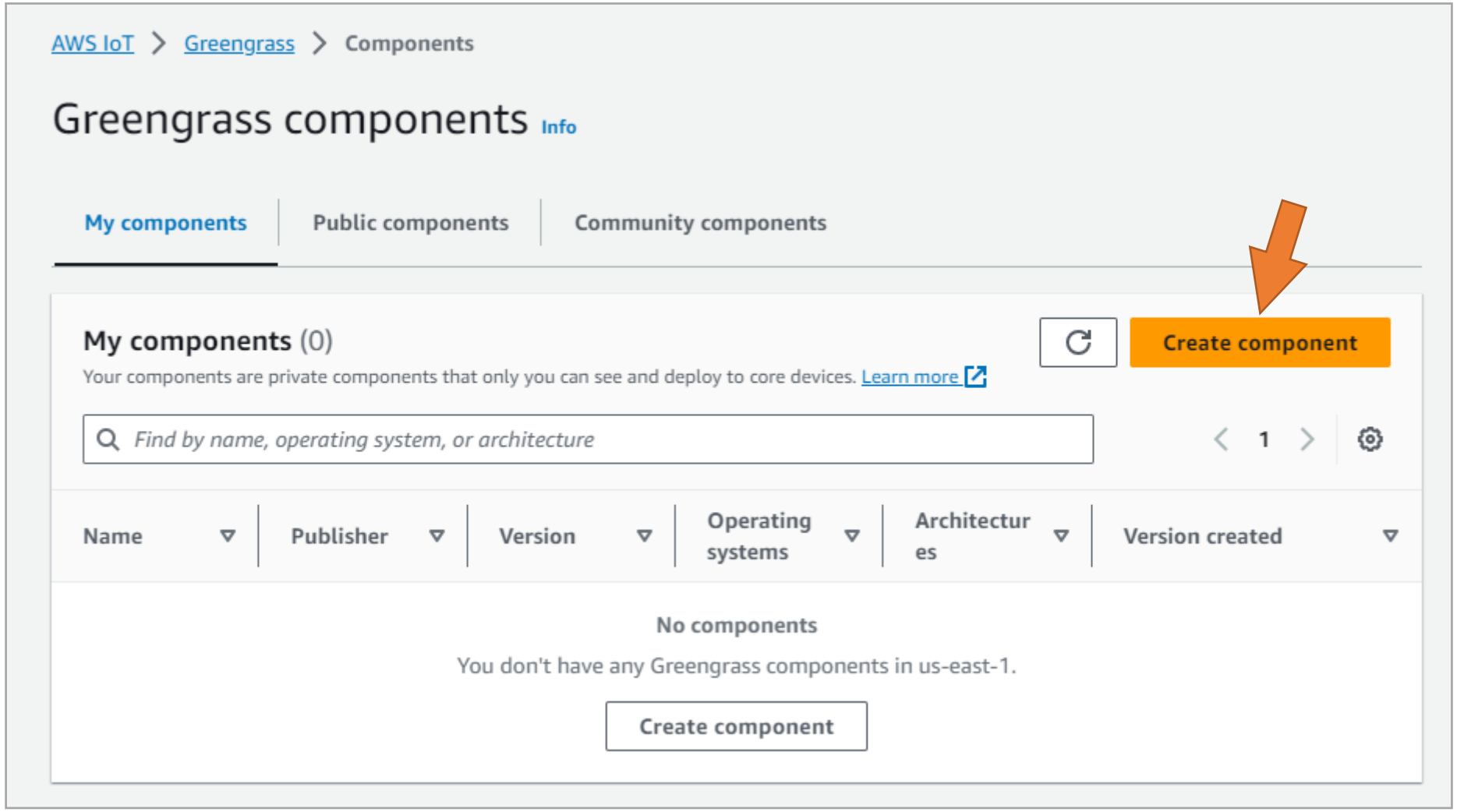


cmpe583-PubSub.py

```
from datetime import datetime
import time
import traceback
import json
import botocore
import sys
import os

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

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

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

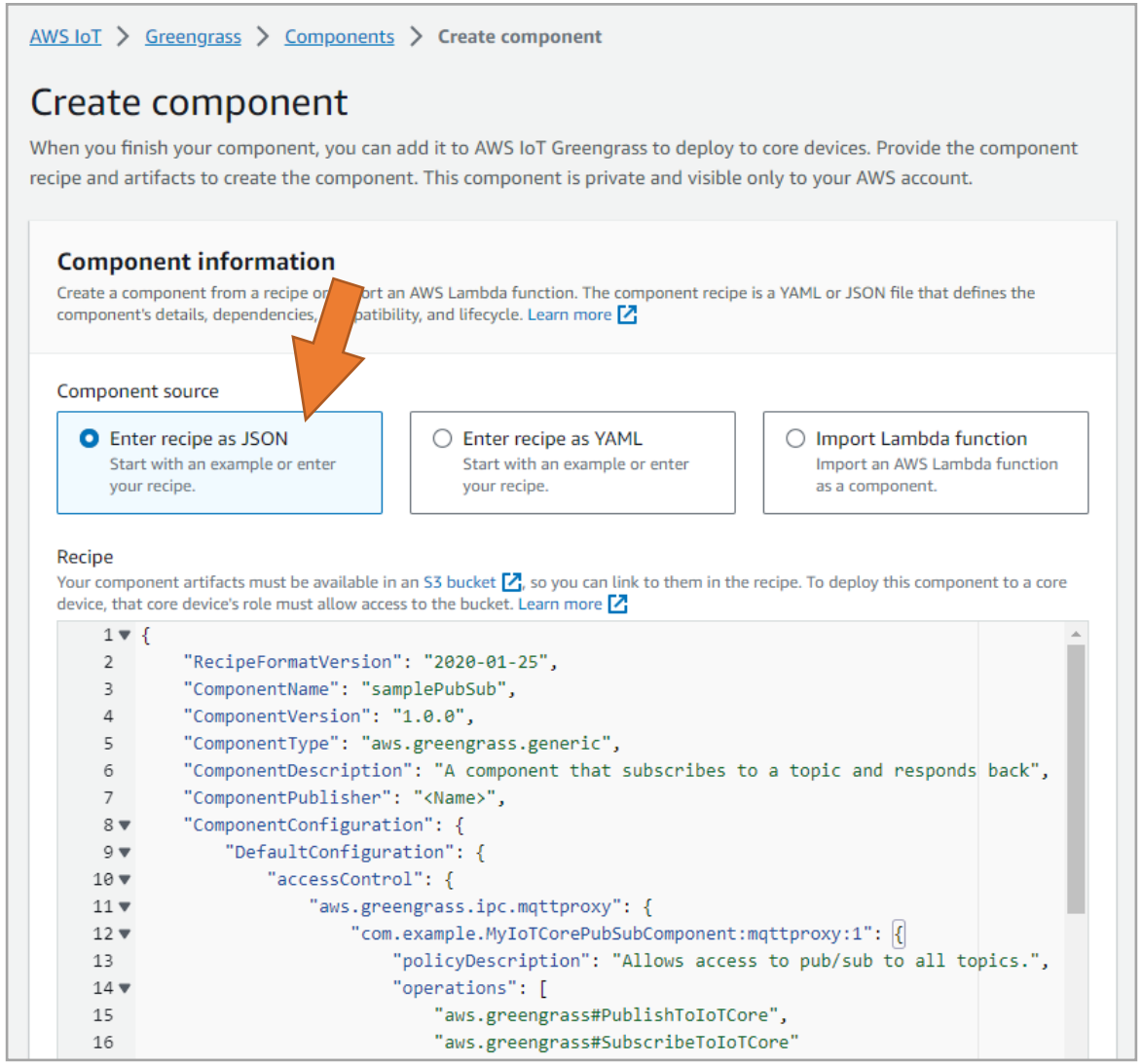
Step 2: Create a Custom Component

Via AWS IoT Greengrass (Console)



Step 2: Create a Custom Component

Enter the Recipe



Step 2: Create a Custom Component

Adjust the Access Control

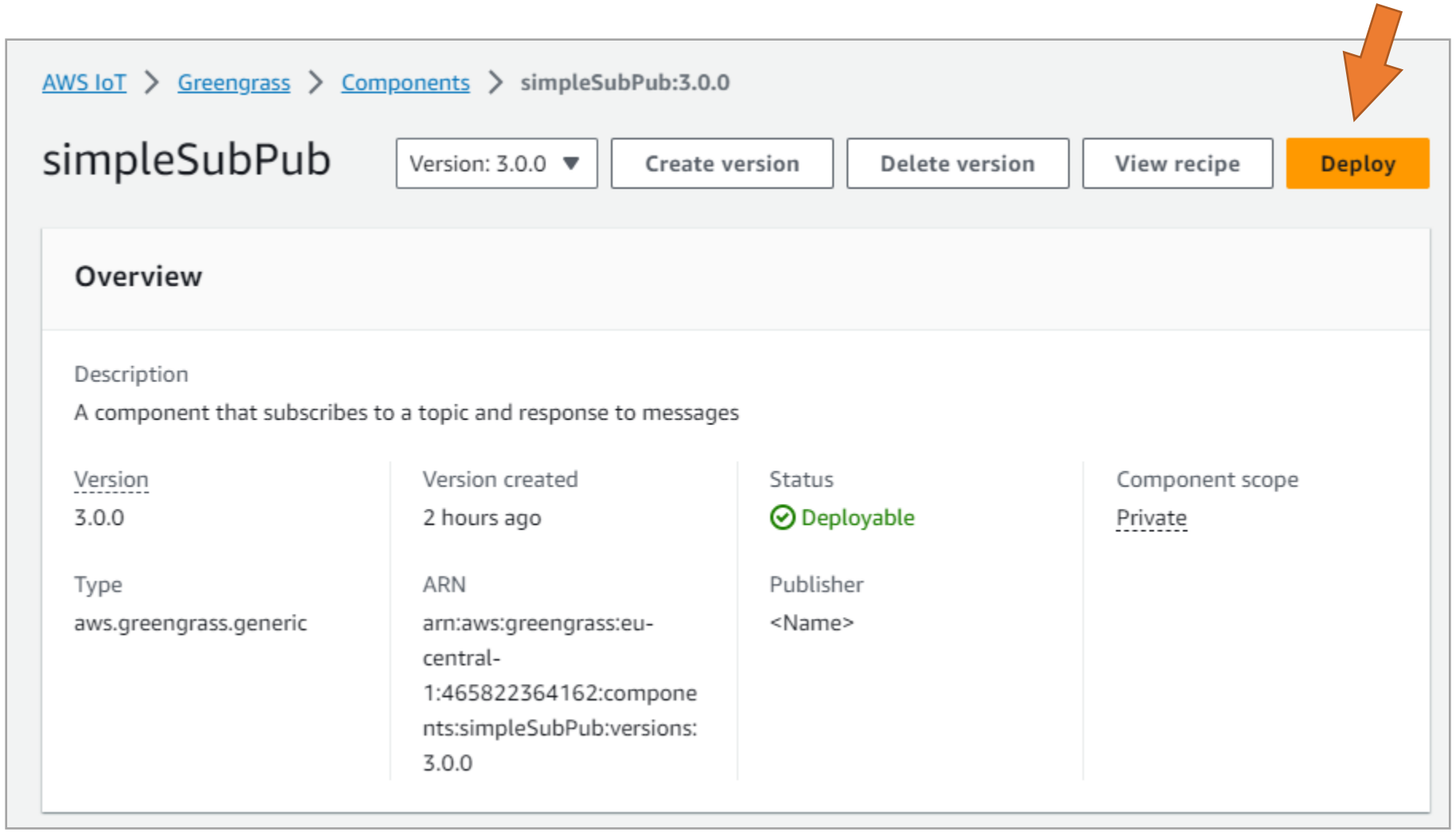
```
{
    "accessControl": {
      "aws.greengrass.ipc.mqttproxy": {
        "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
          "policyDescription": "Allows access to publish/subscribe to all topics.",
          "operations": [
            "aws.greengrass#PublishToIoTCore",
            "aws.greengrass#SubscribeToIoTCore"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

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

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

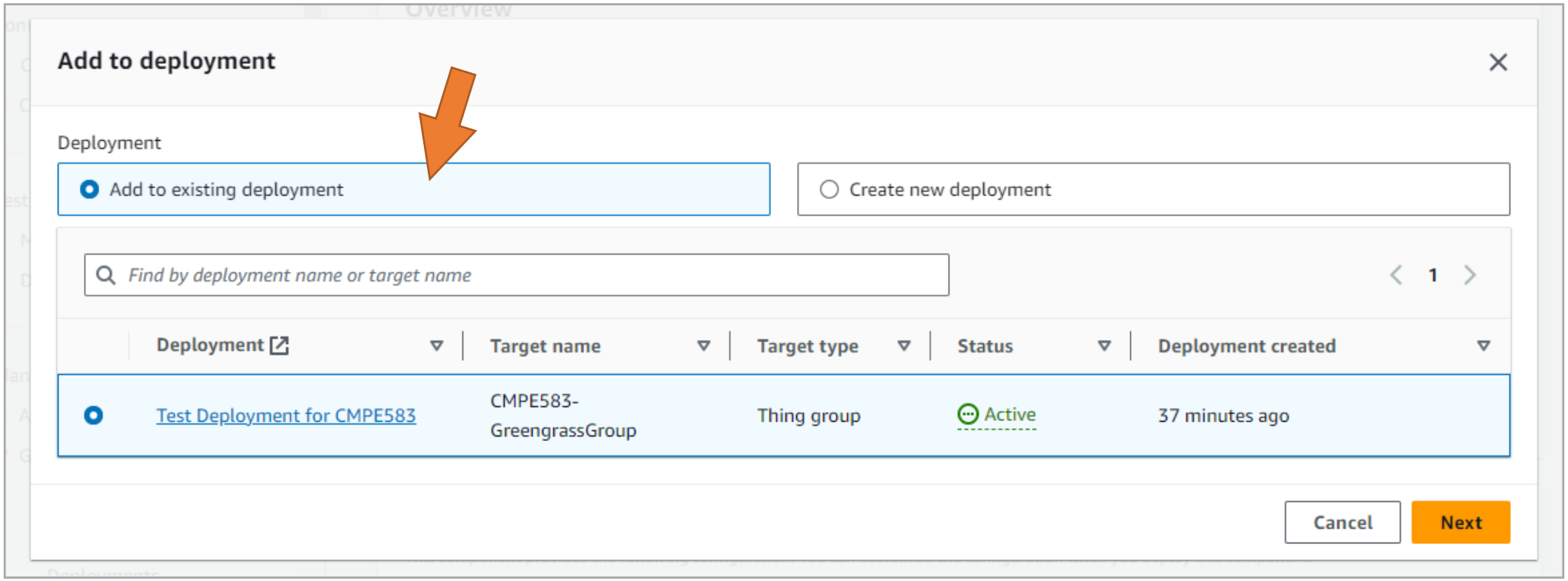
Step 3: Deploy Your Component

Via AWS IoT Greengrass (Console)



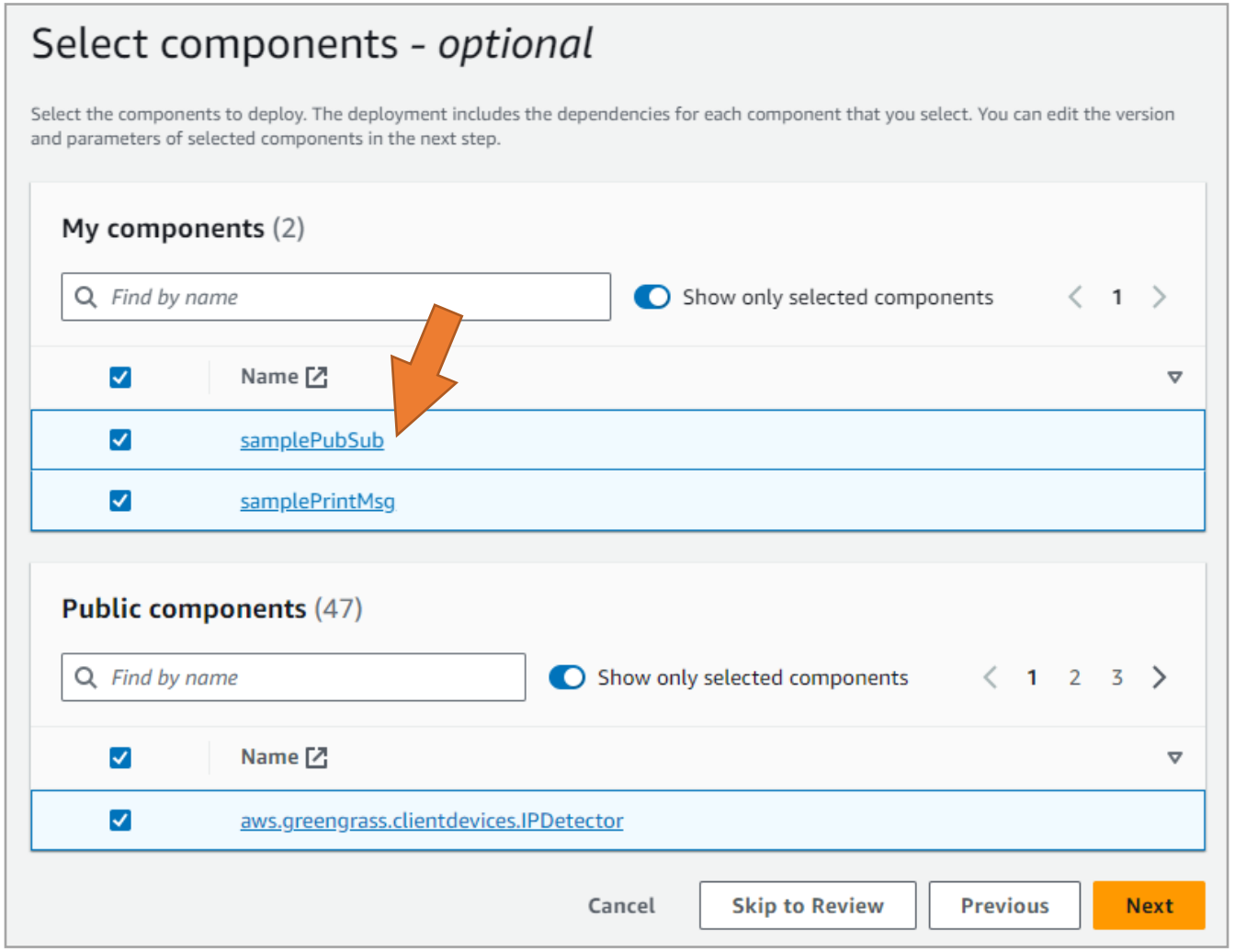
Step 3: Deploy Your Component

Add Custom Component to Existing Deployment



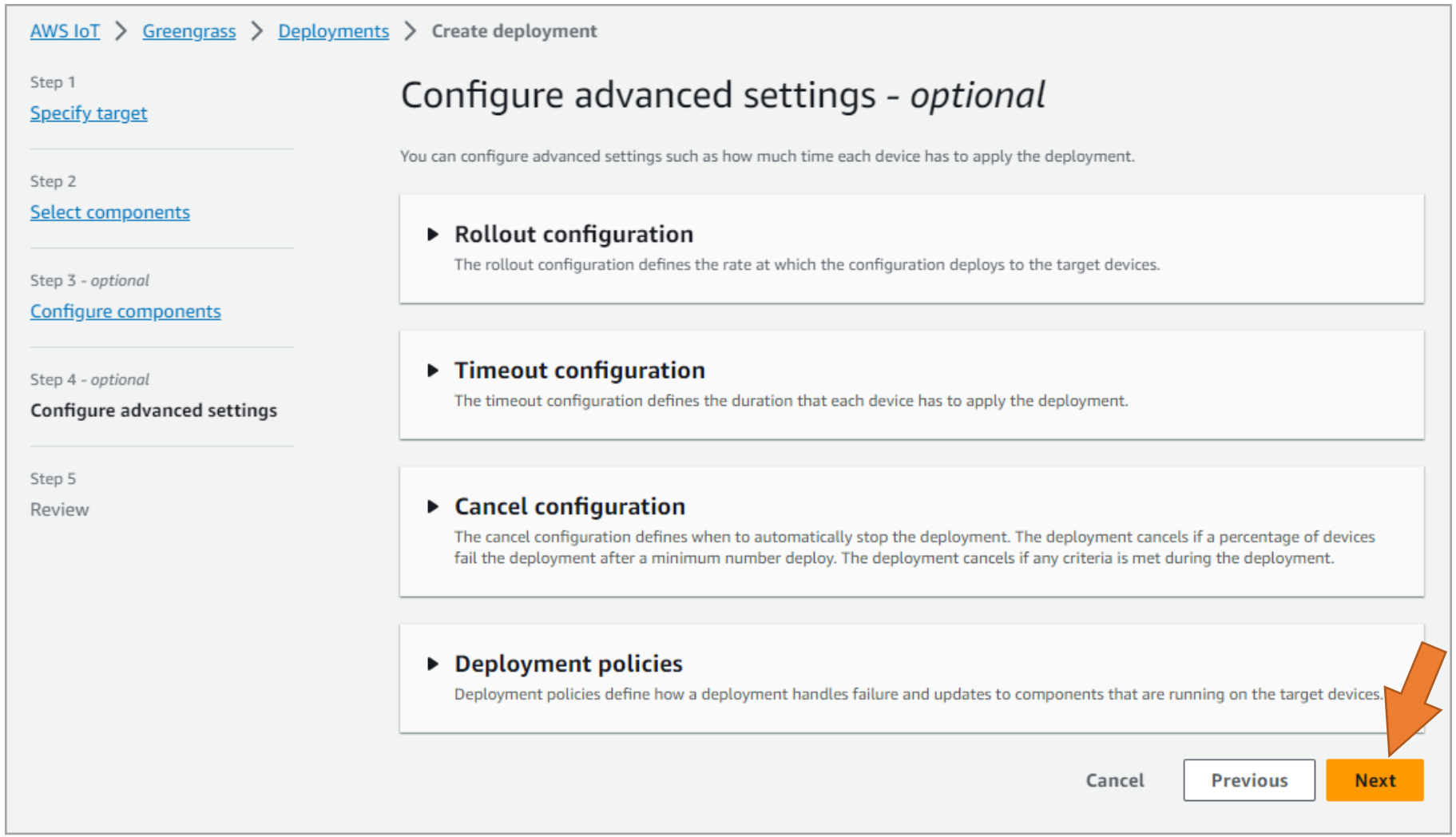
Step 3: Deploy Your Component

Select the Component



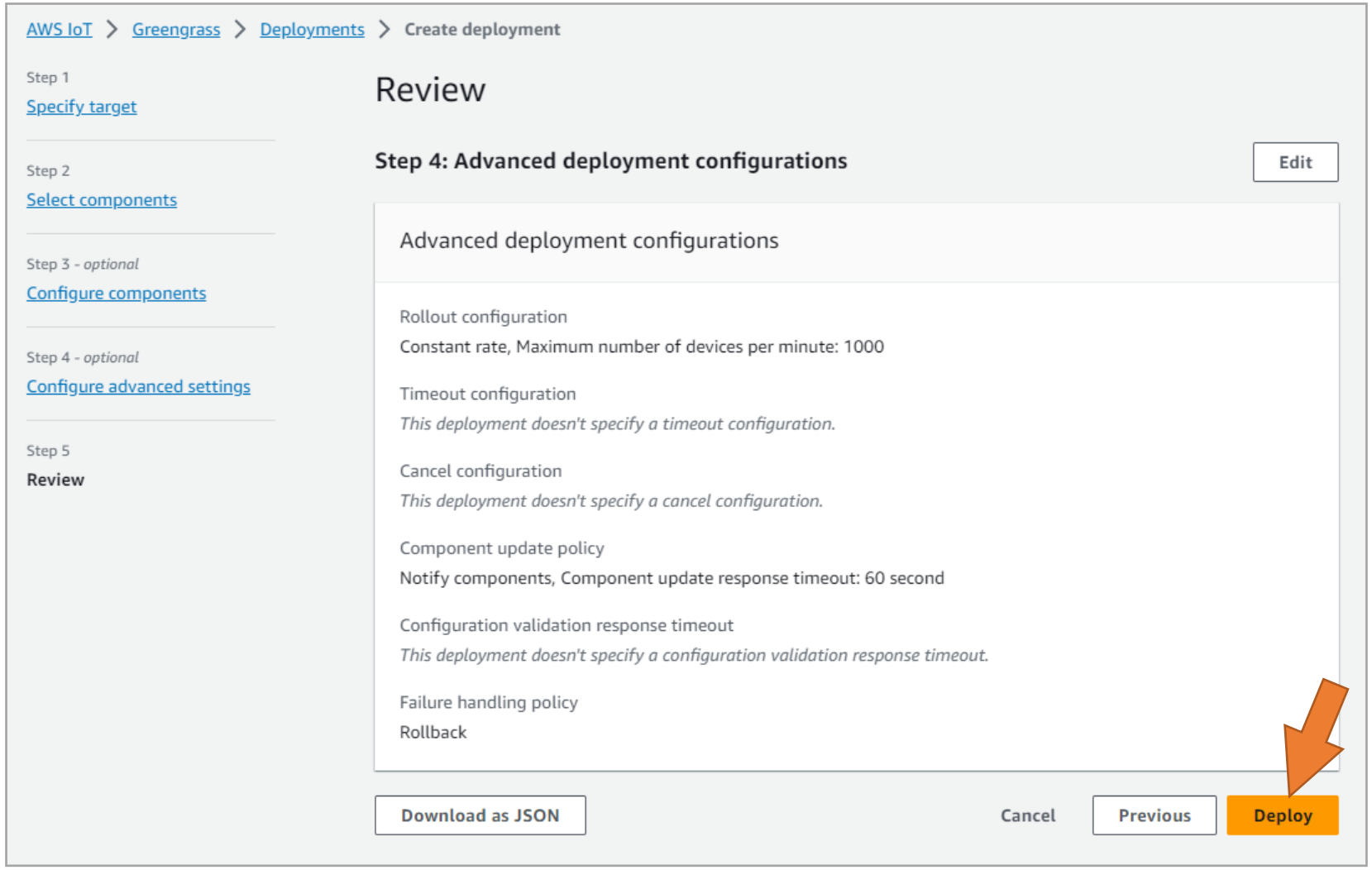
Step 3: Deploy Your Component

Configure Advanced Settings



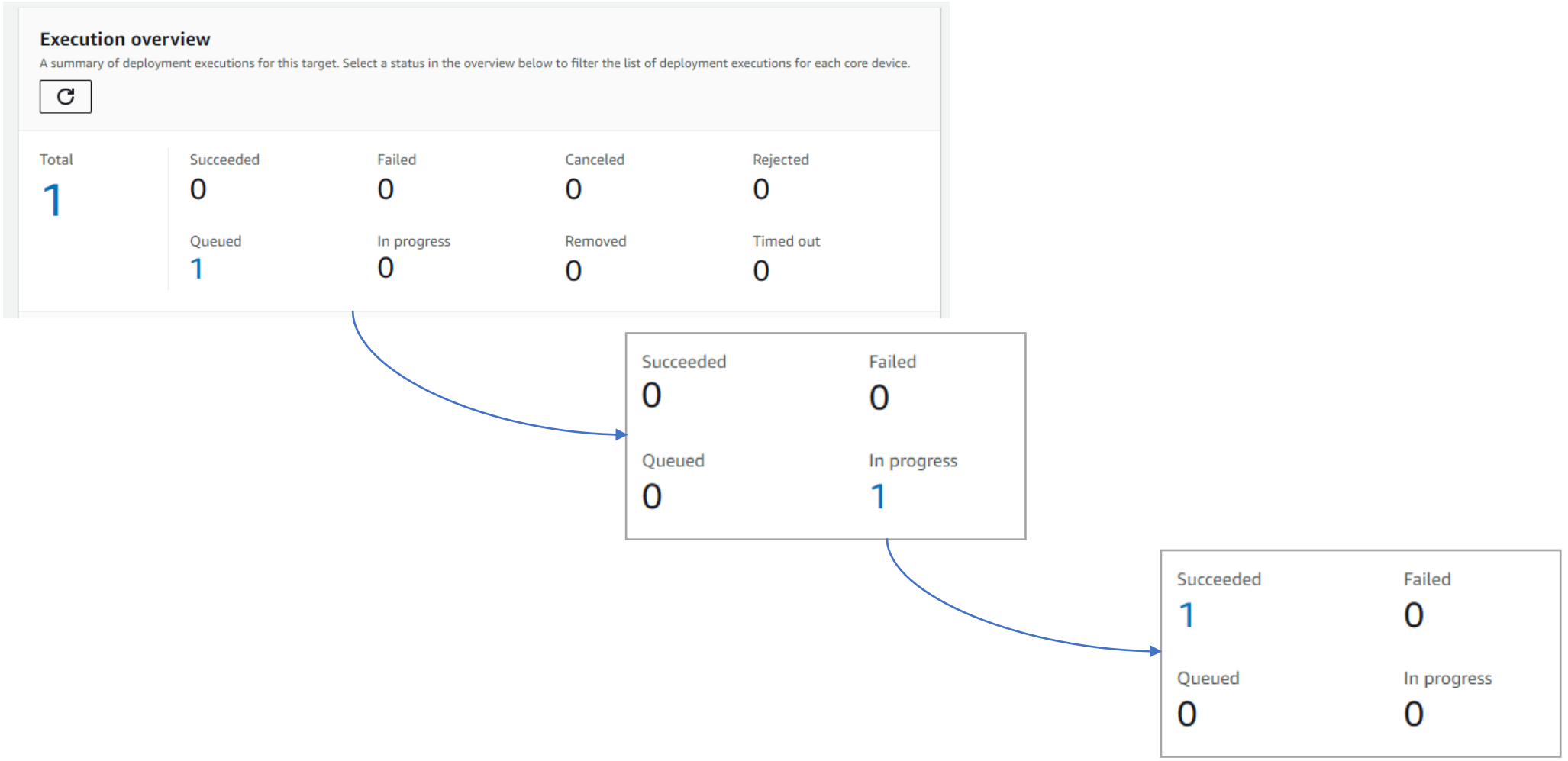
Step 3: Deploy Your Component

Review & Deploy



Step 4: Verify the Component

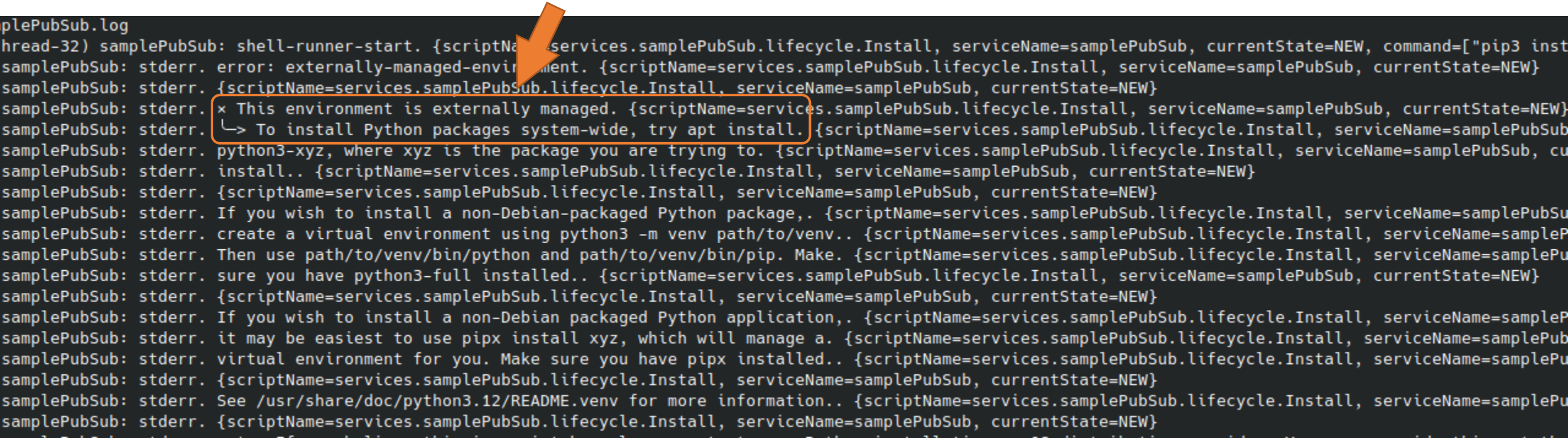
Check AWS IoT Greengrass Console



Troubleshooting

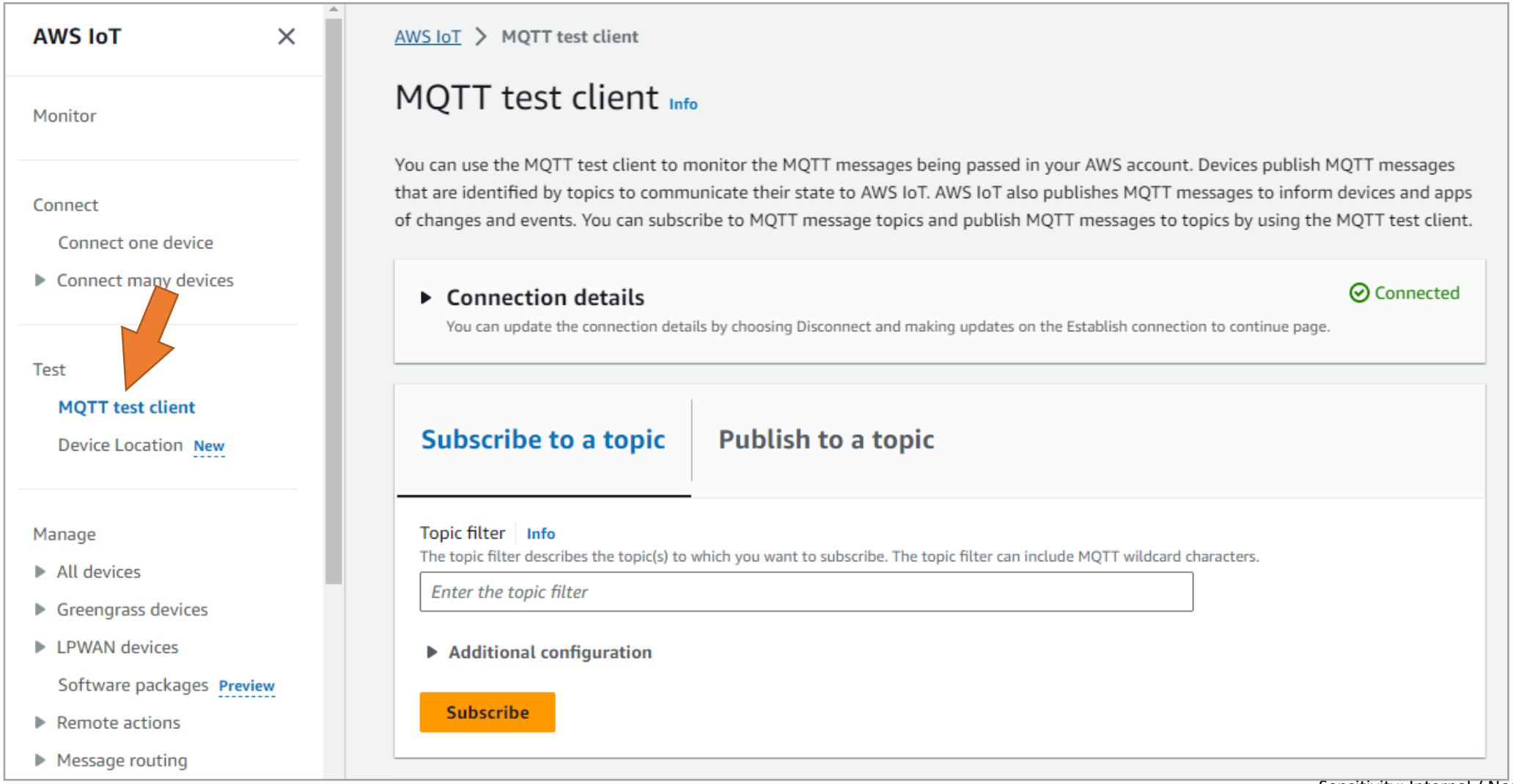
Check greengrass.log File

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



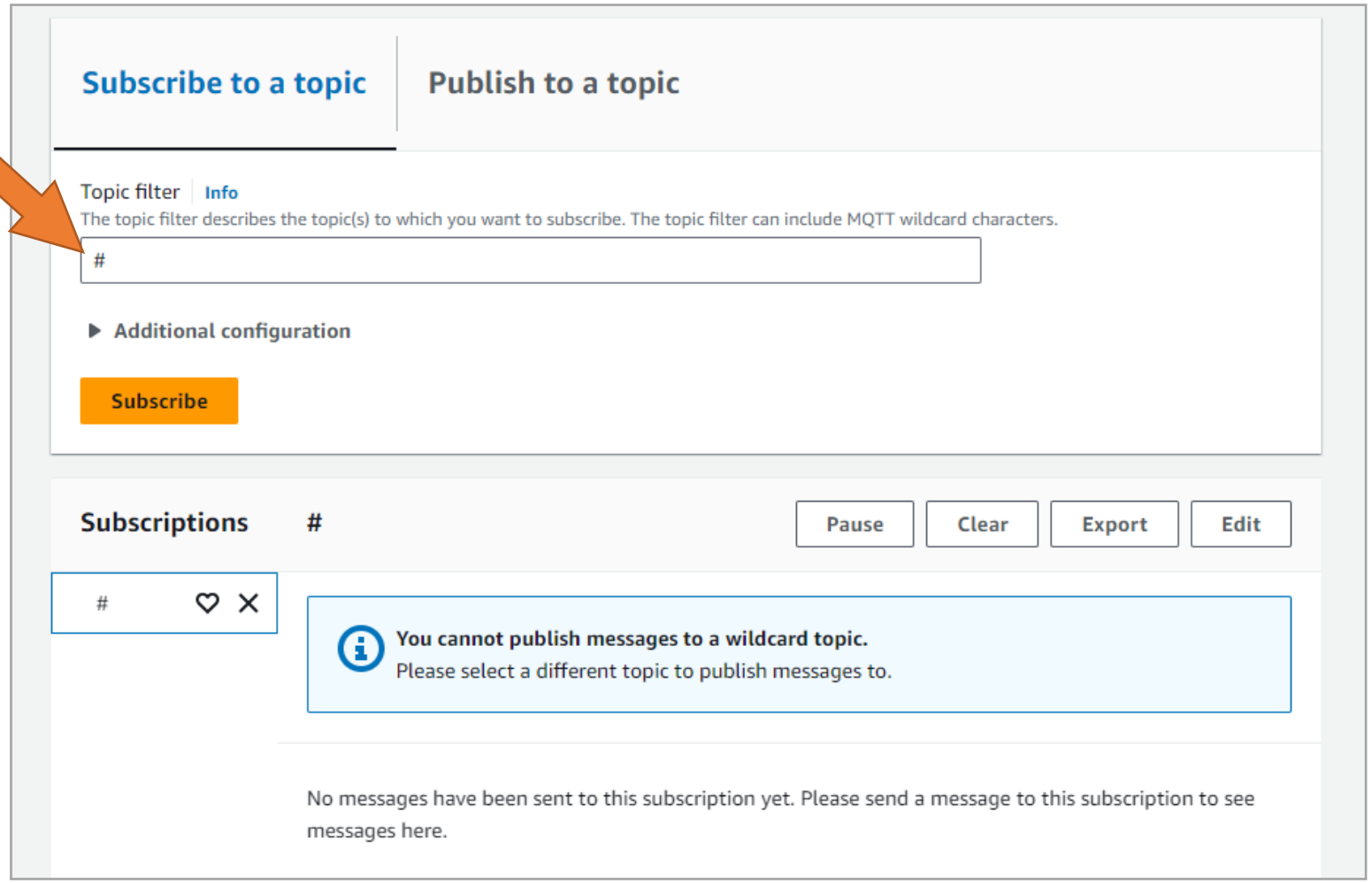
Test Your Deployment

Via AWS MQTT Test Client



Test Your Deployment

Subscribe All Topics

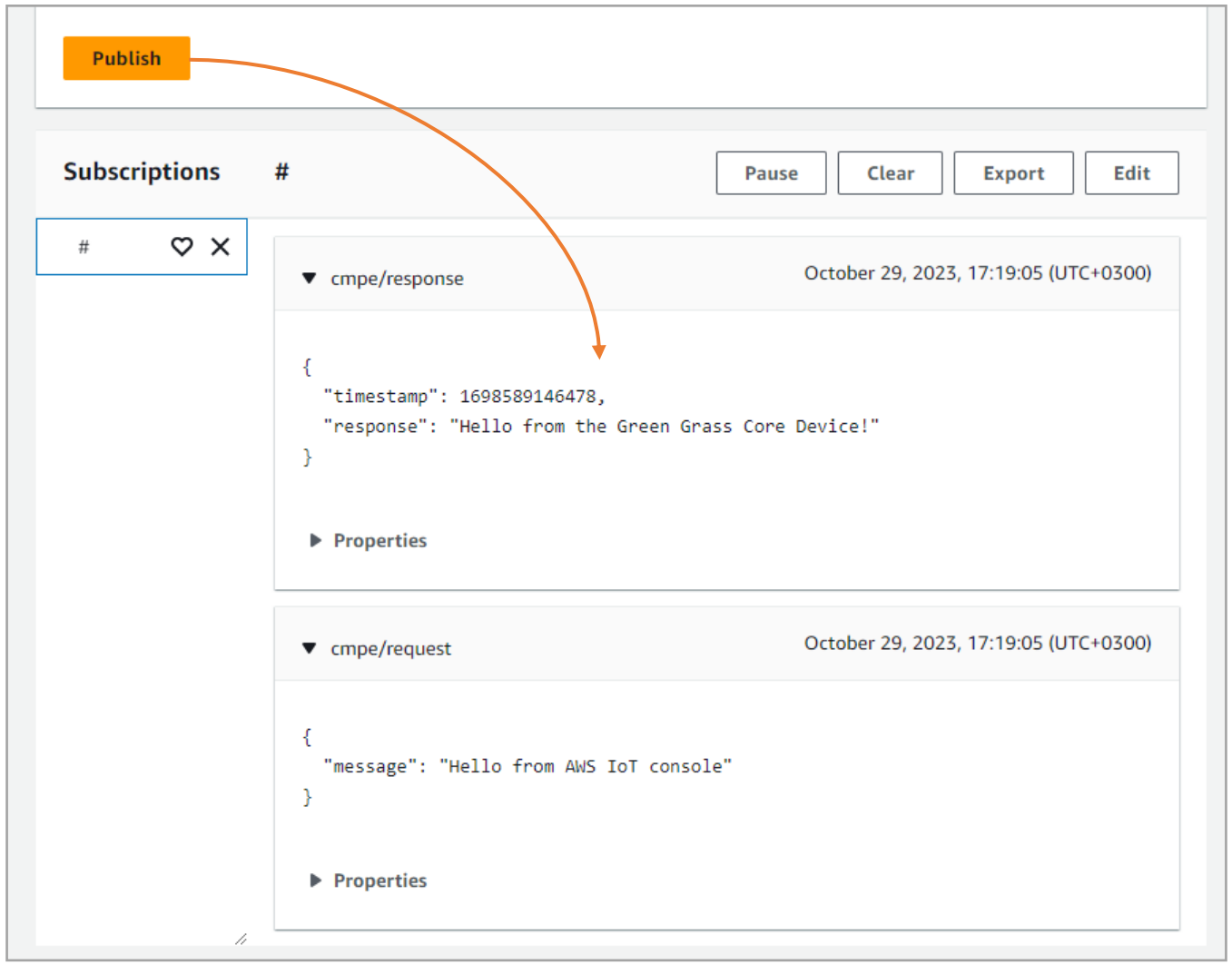


Test Your Deployment

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

AWS Lambda Functions on Core Devices

Run AWS Lambda Functions on Core Devices

- If you want to deploy an existing 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

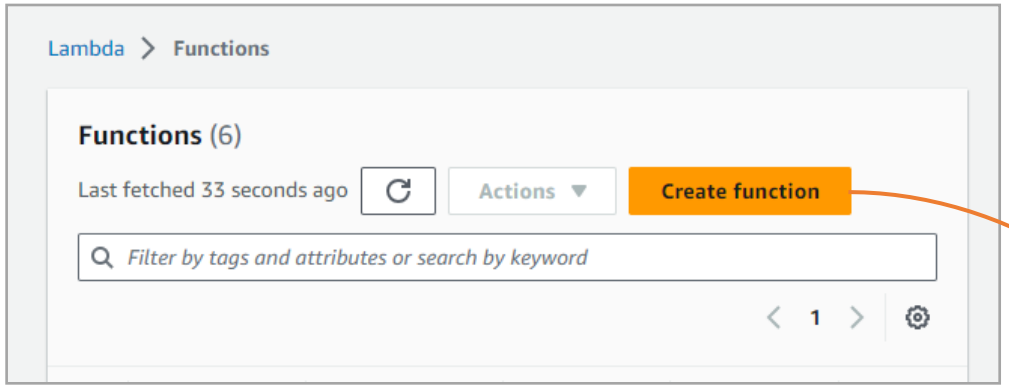
- On-demand functions start when they are invoked and stop when there are no tasks left to run.
- Each invocation of the function creates a separate container, also called a sandbox, to process invocations, unless an existing container is available for reuse.
- Any of the containers might process data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

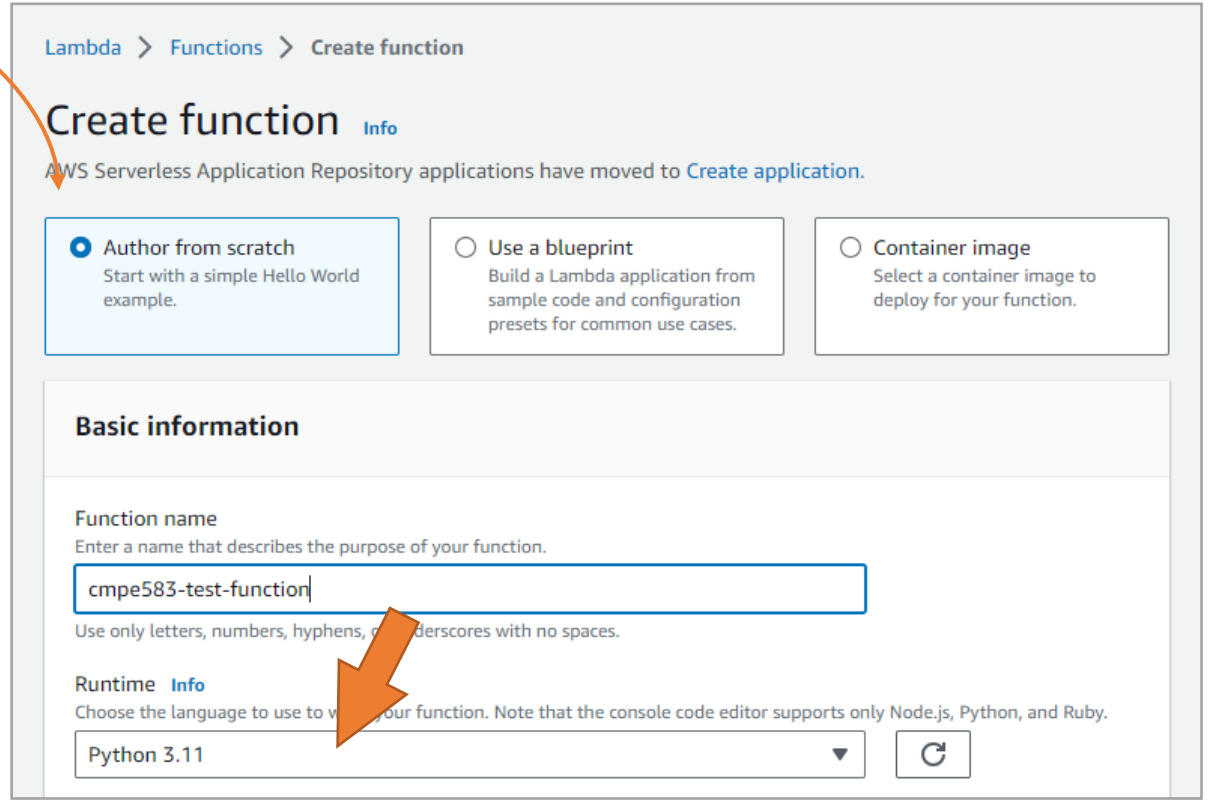
Long-lived functions

- 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.

Step 1: Create AWS Lambda Function

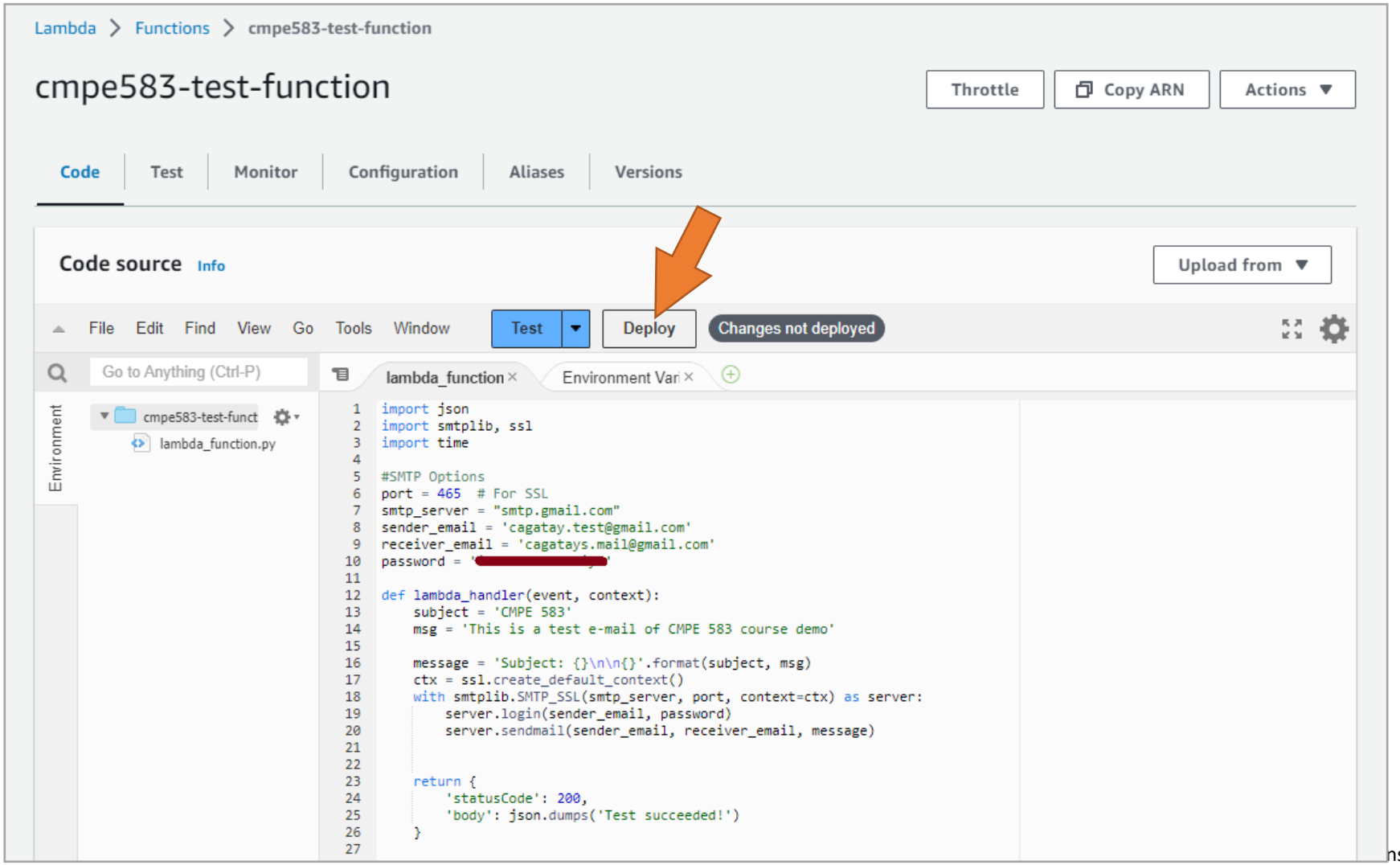
Via AWS Lambda (Console)





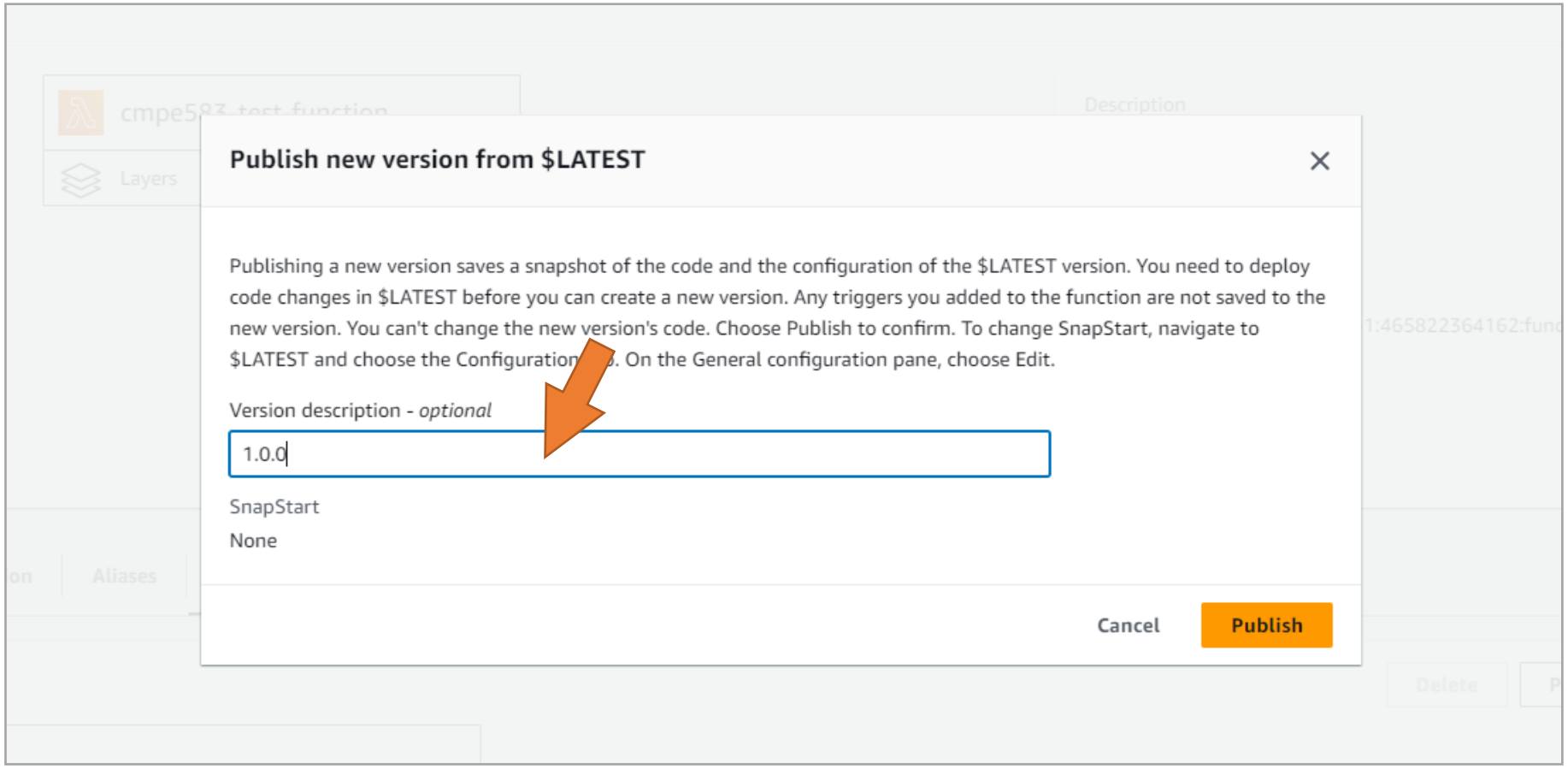
Step 1: Create AWS Lambda Function

Deploy the Lambda Function



Step 1: Create AWS Lambda Function

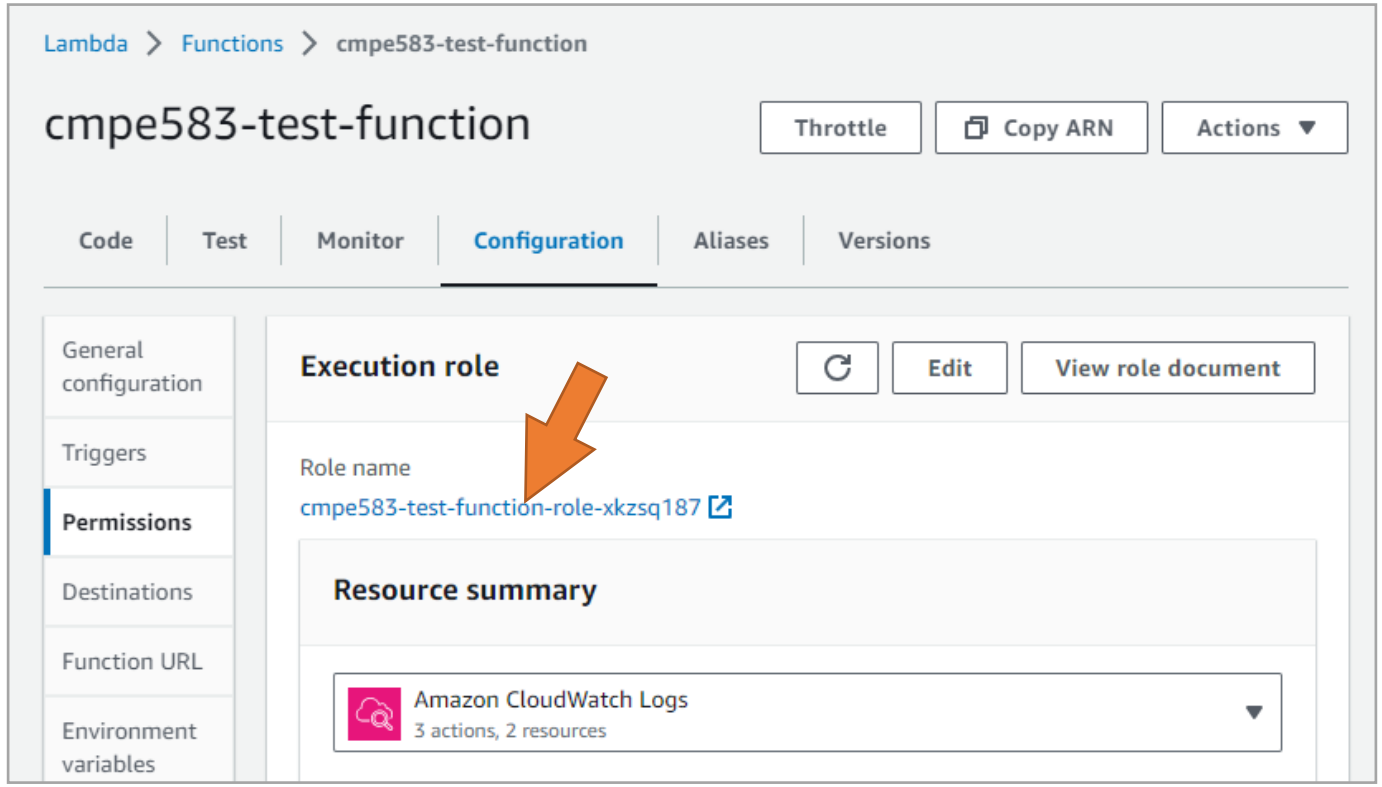
Publish the First Version



Step 1: Create AWS Lambda Function

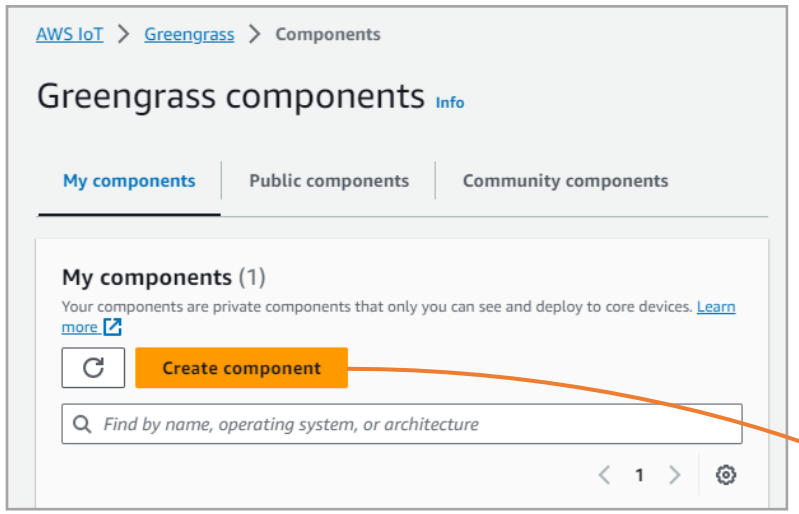
Configure Lambda Role

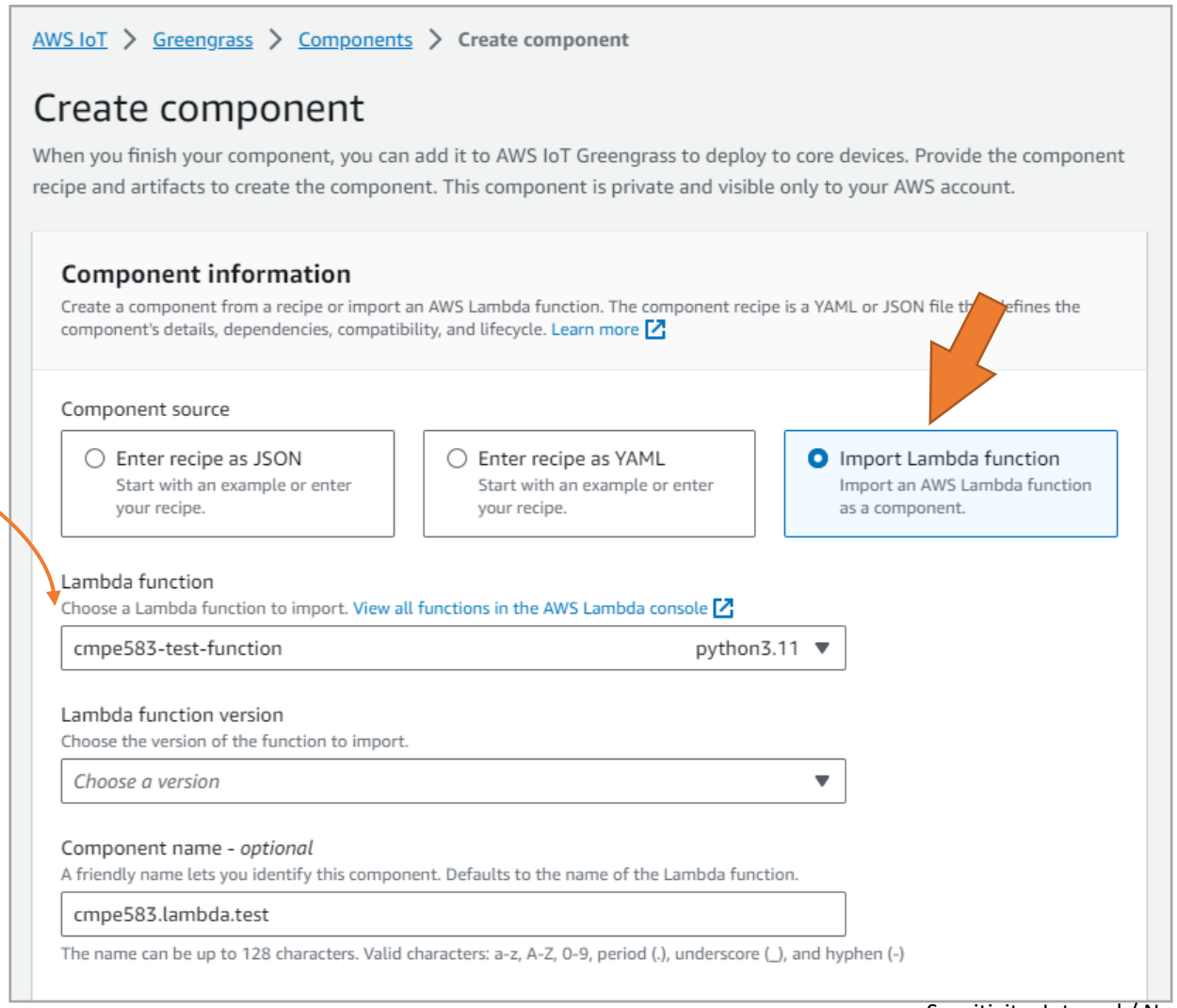
- Our lambda function is very simple for testing purpose, but if you have a more complex function that needs to access AWS resources (server, S3 file, database, etc.), you will need to configure the lambda's role accordingly.



Step 2: Create a Custom Component

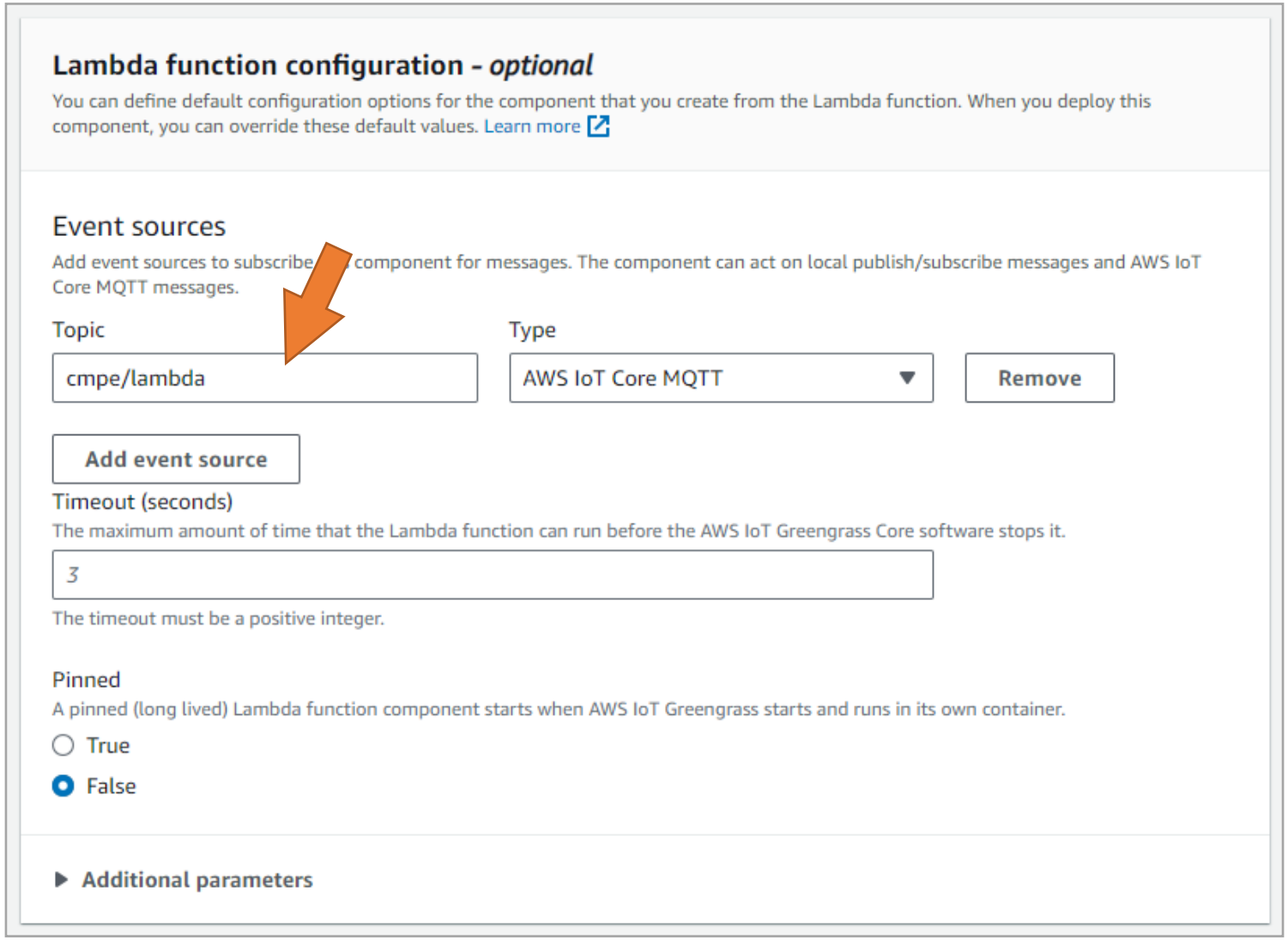
Import Lambda Function





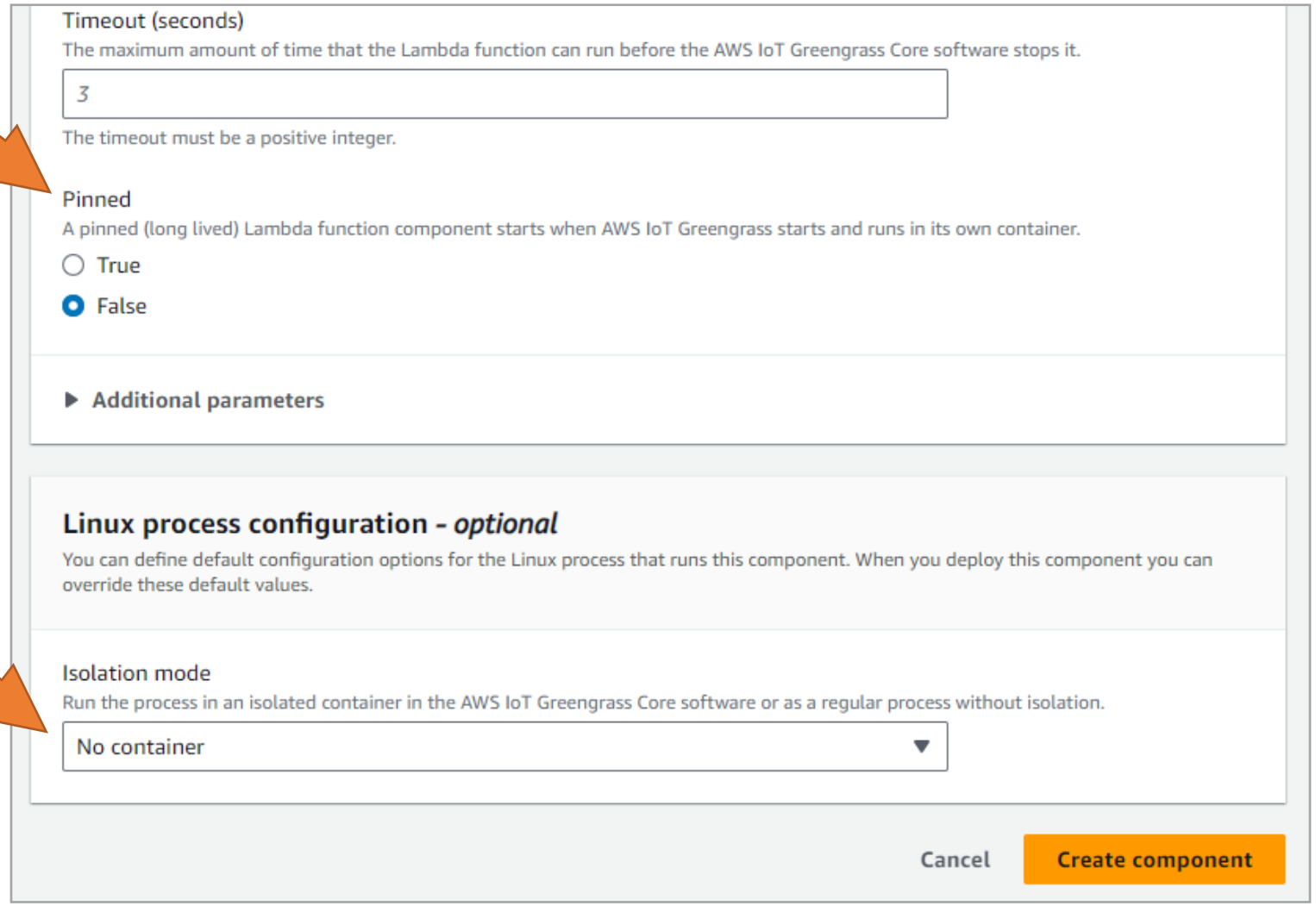
Step 2: Create a Custom Component

Configure Event Source to Trigger Lambda Function



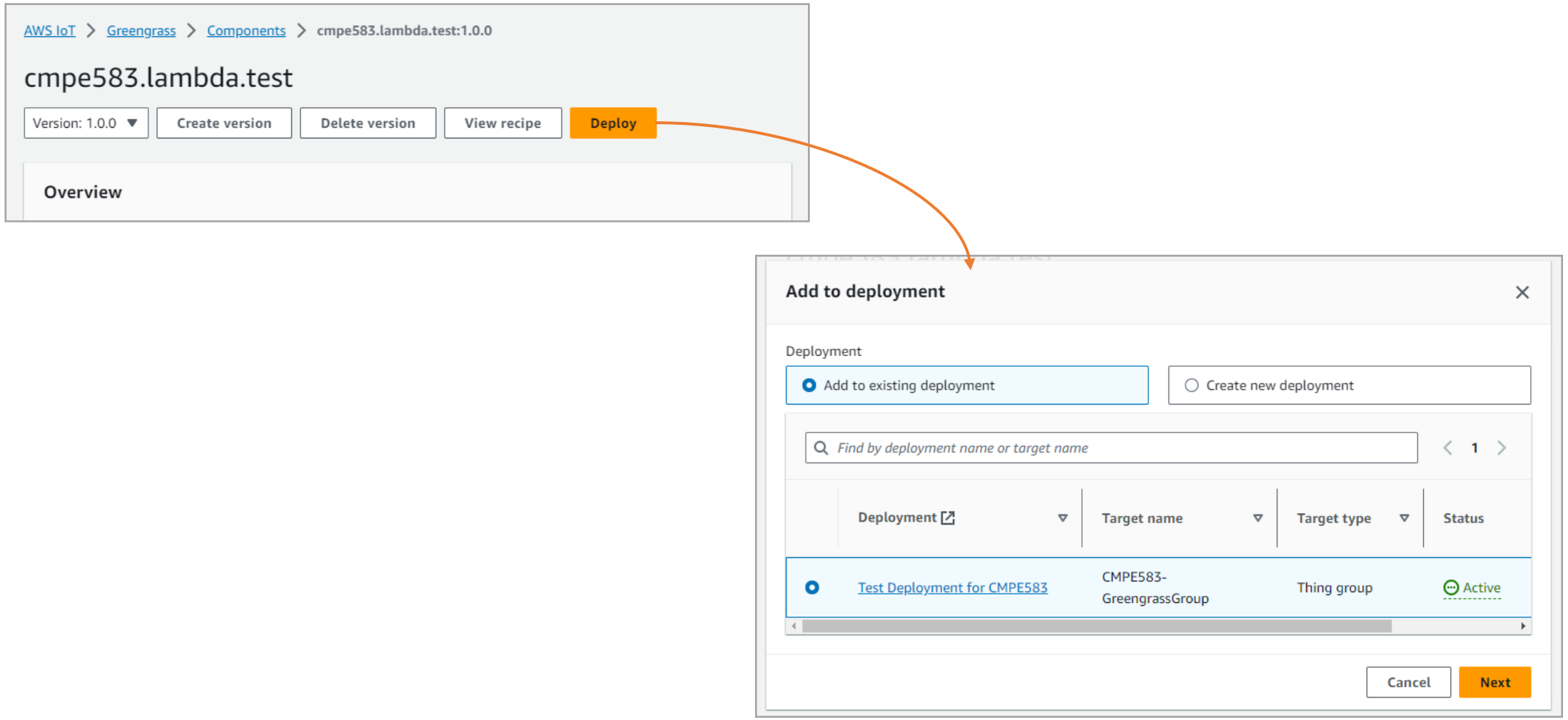
Step 2: Create a Custom Component

Configure Lambda Lifecycle



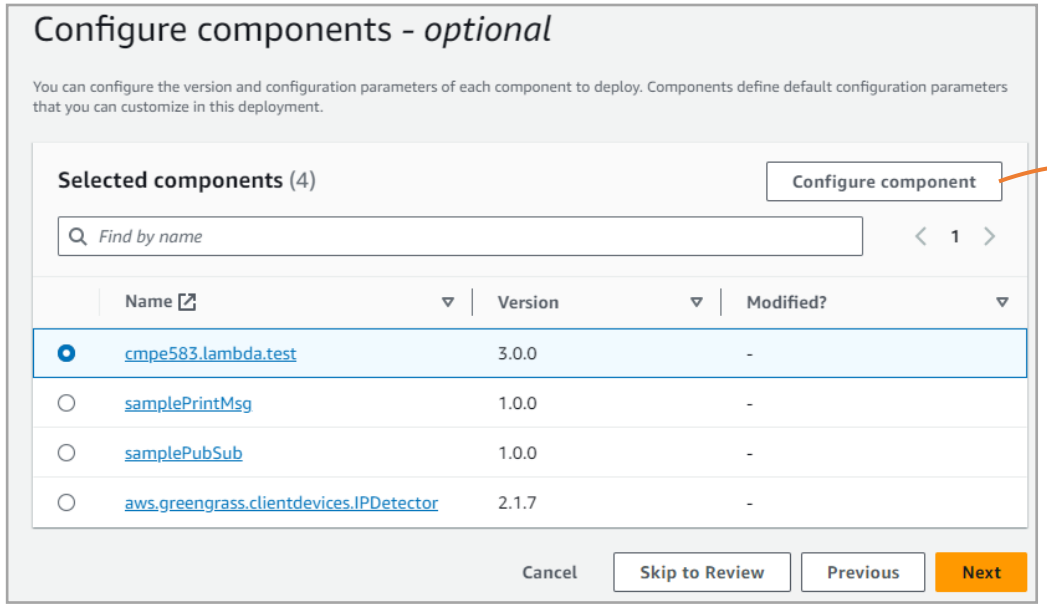
Step 3: Deploy Your Component

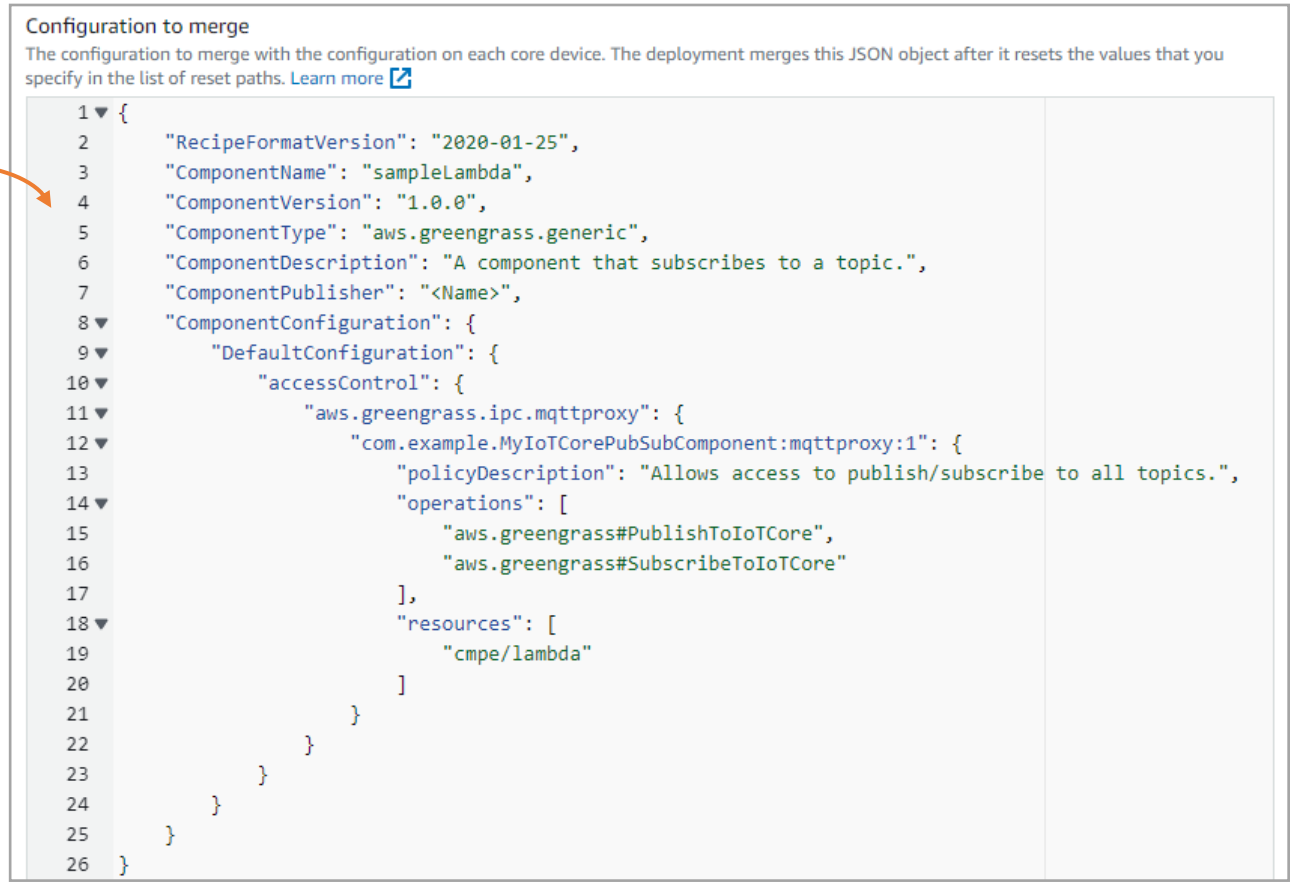
Add Component to Existing Deployment



Step 3: Deploy Your Component

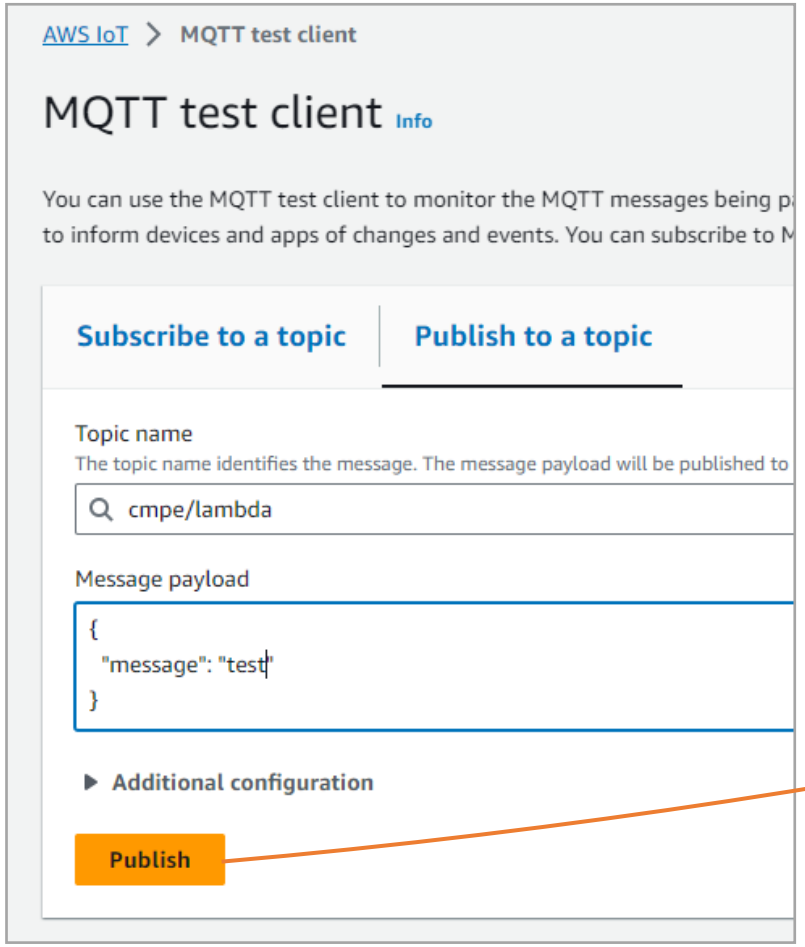
Configure Component to Merge Recipe

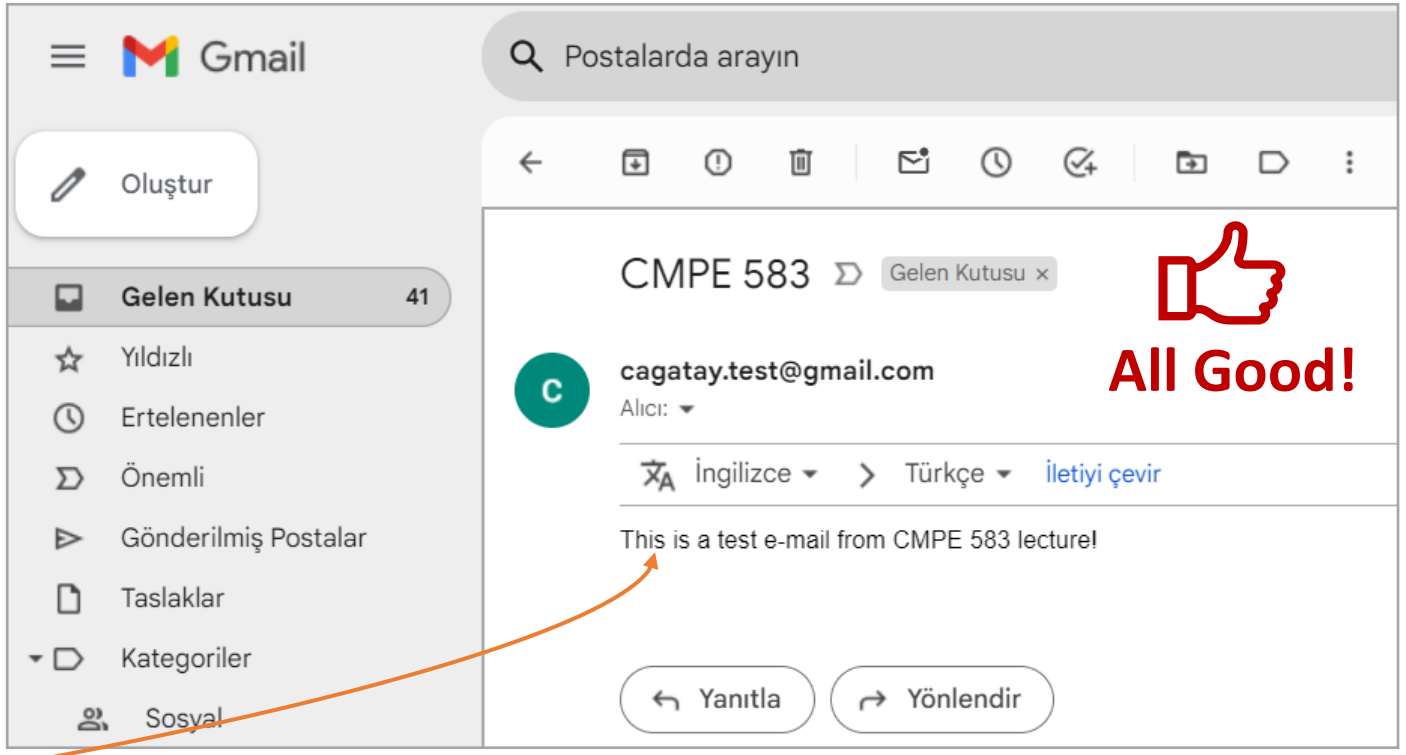




Test Your Deployment

Via AWS MQTT Test Client





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}
```

Local IoT (Client) Devices

Local IoT Devices

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

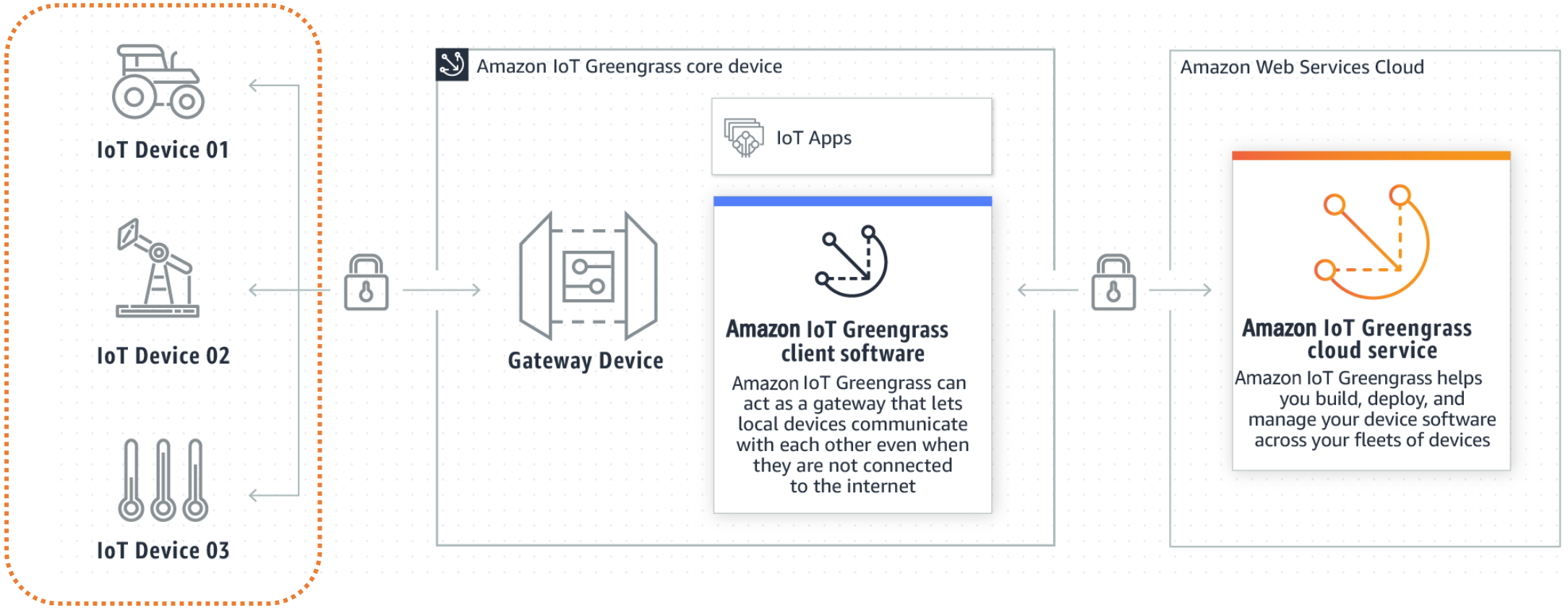


image source: https://aws.amazon.com/greengrass/

Client Device and Core Device Communication

Use -Cases

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

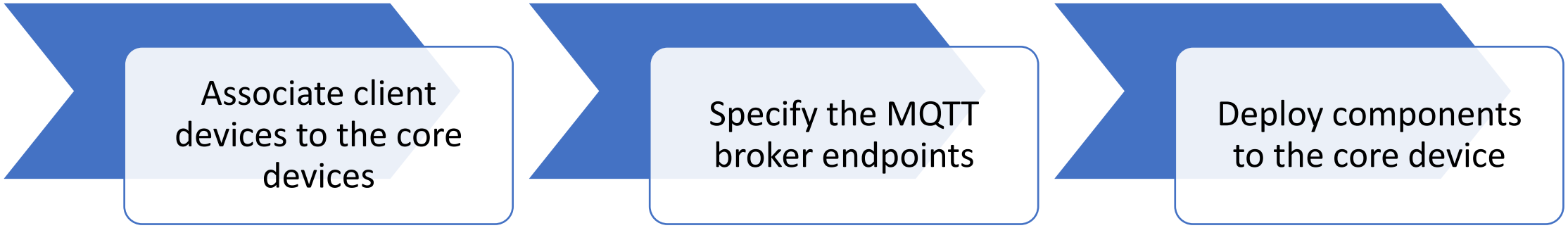
AWS-Provided Client Device Components

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

ata

Cloud Discovery Configuration

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

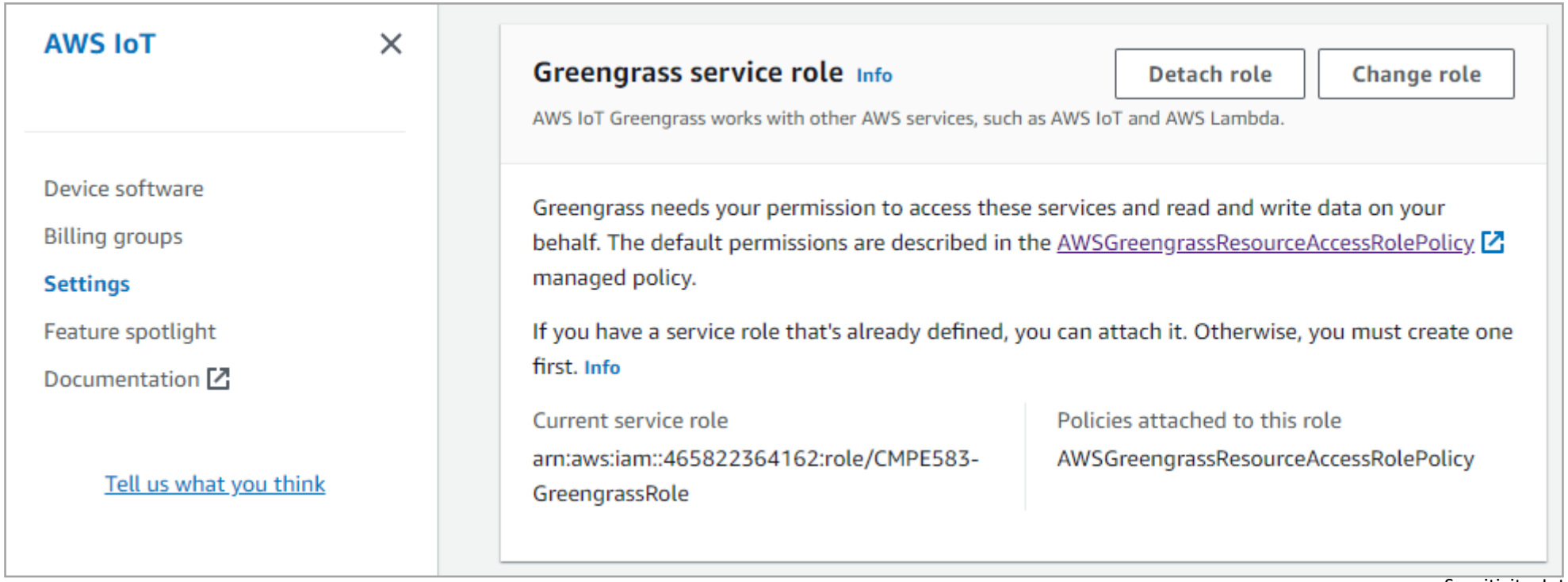


Create Client Devices

Step 1: Configure the Greengrass Service Role

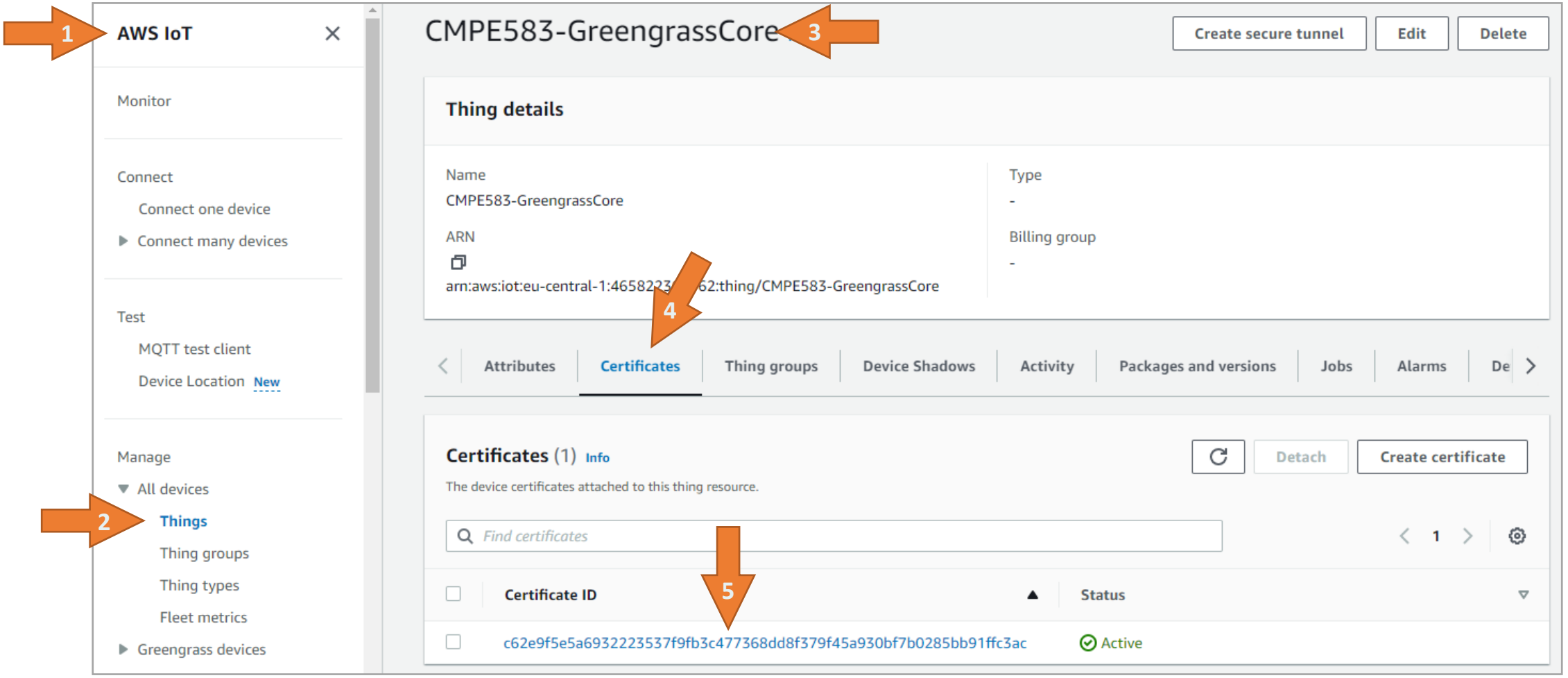
Verify If Greengrass Service Role is Attached

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



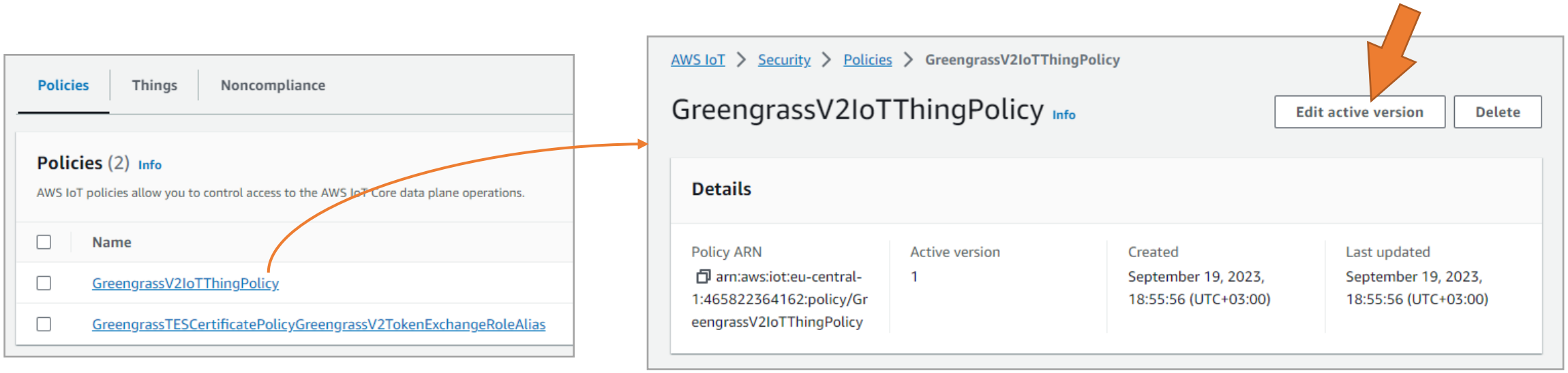
Step 2: Configure the AWS IoT Thing Policy

Via AWS IoT Console



Step 2: Configure the AWS IoT Thing Policy

Review the Policy

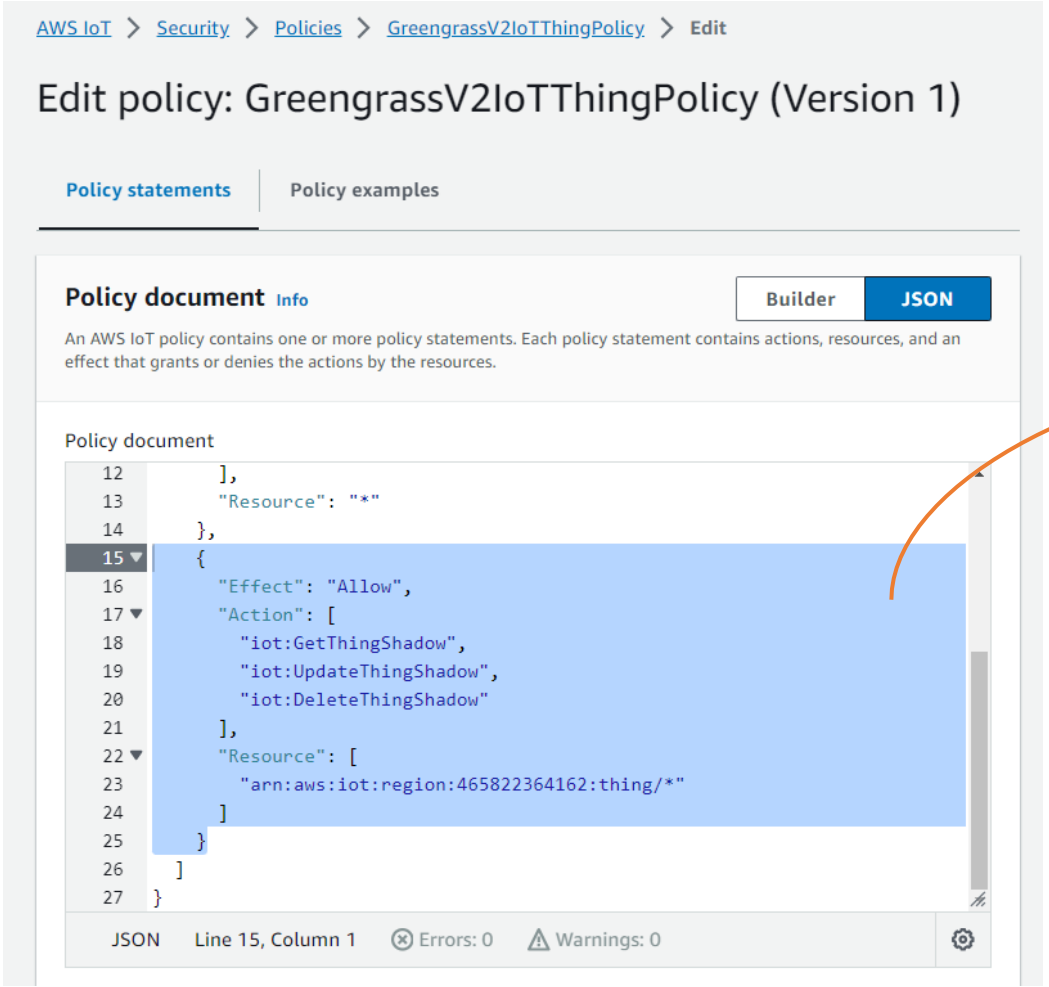


➢ Review the policy for required permissions, and add any required permissions that are missing:

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

Step 2: Configure the AWS IoT Thing Policy

Allow Core Device to Sync Shadows

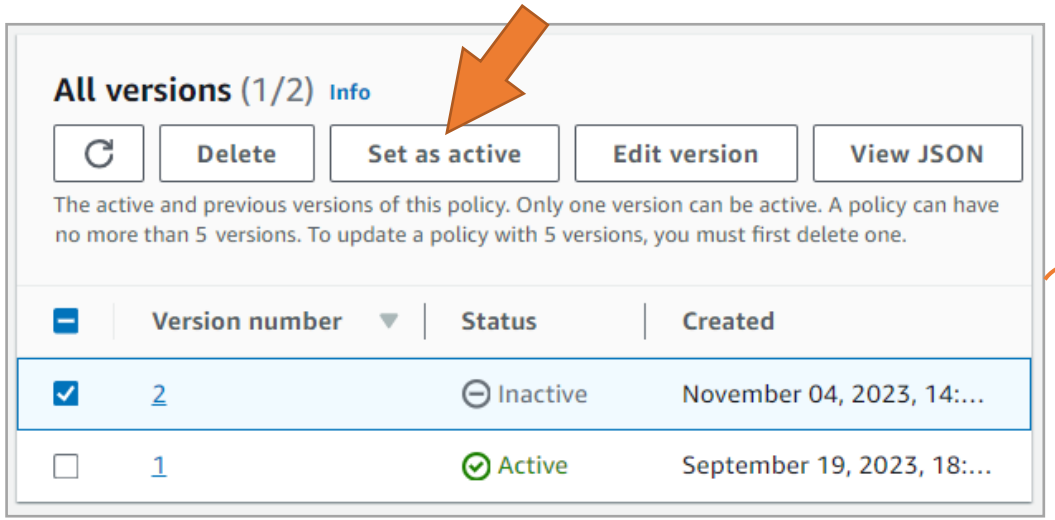


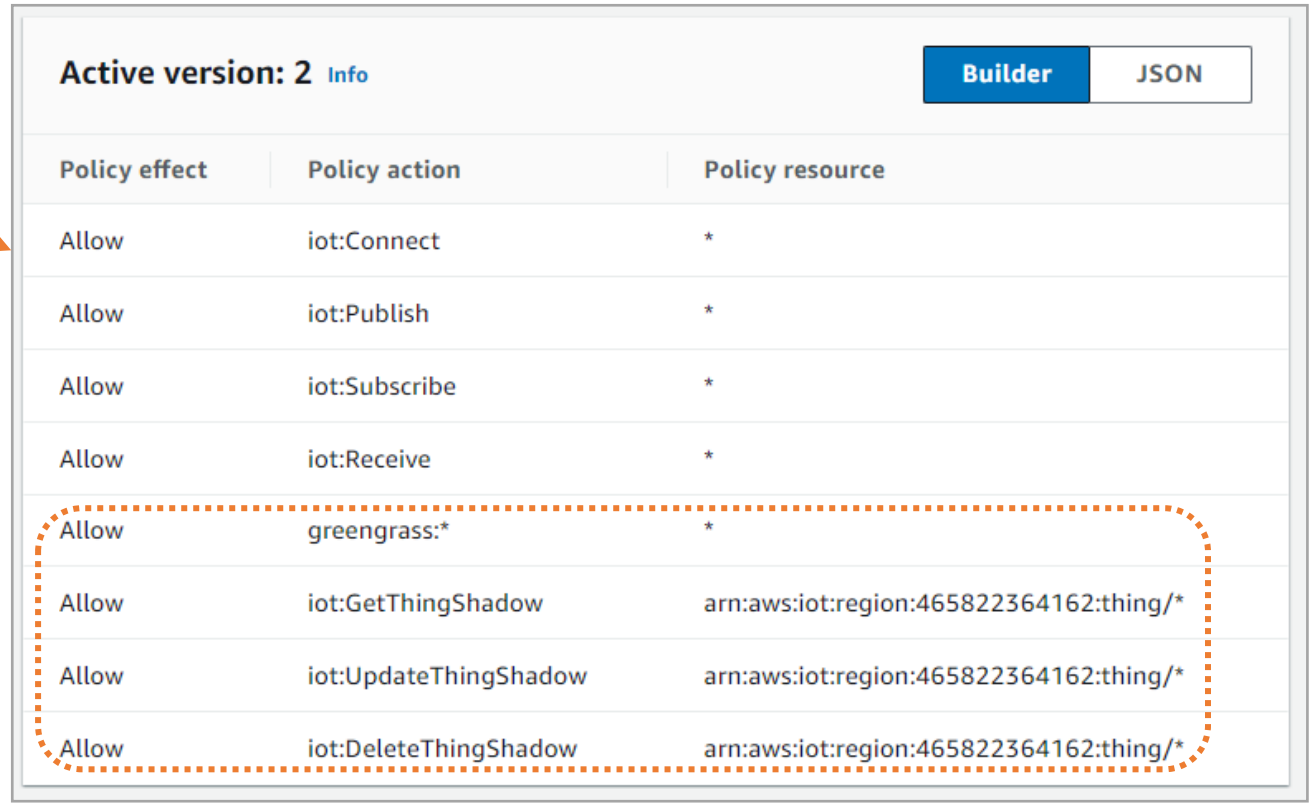
To allow the core device to sync shadows with AWS IoT Core, add the following statement to the policy:

```
{
  "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

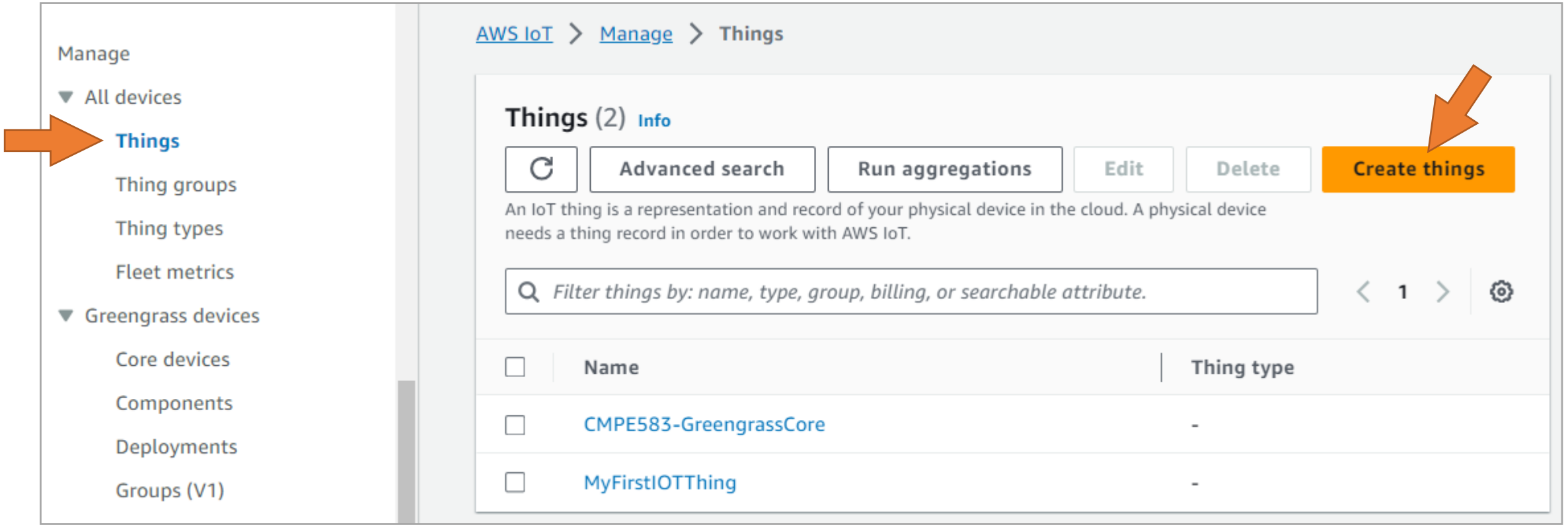
Update & Verify Policy





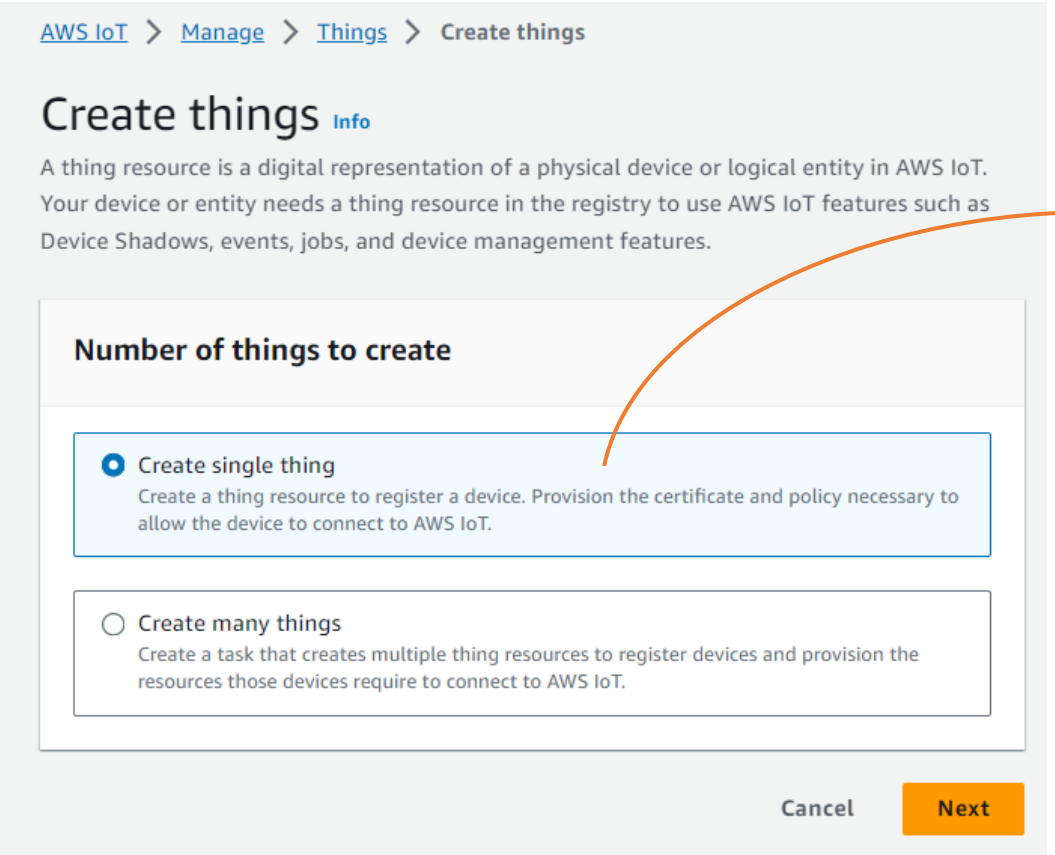
Step 3: Create Client Device (Thing)

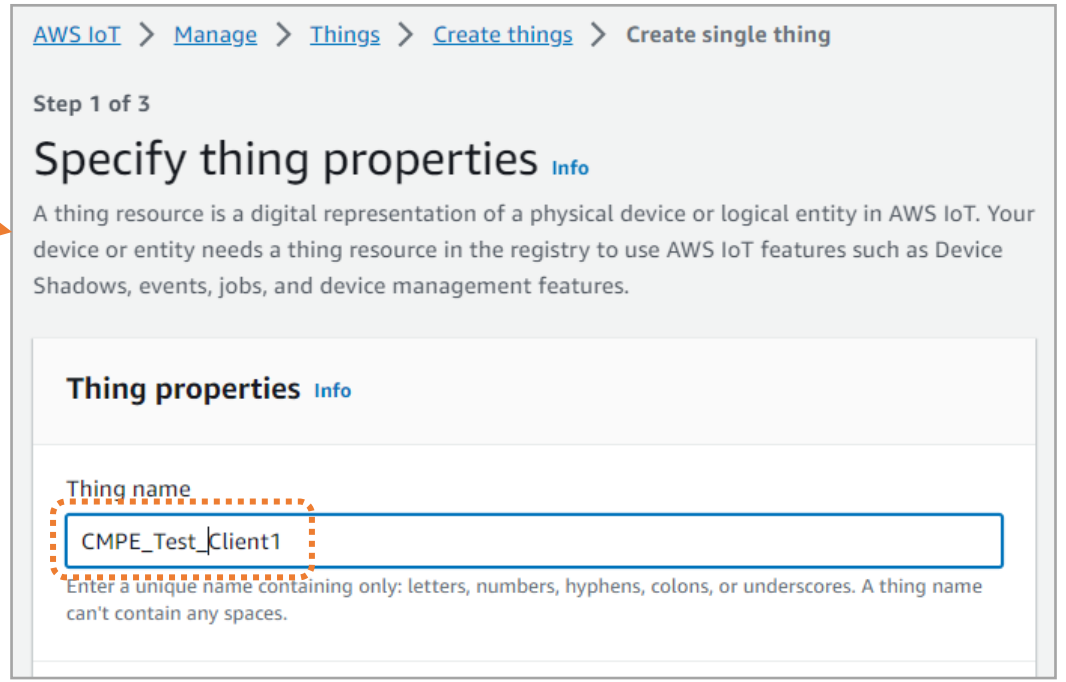
Via AWS IoT Console



Step 3: Create Client Device (Thing)

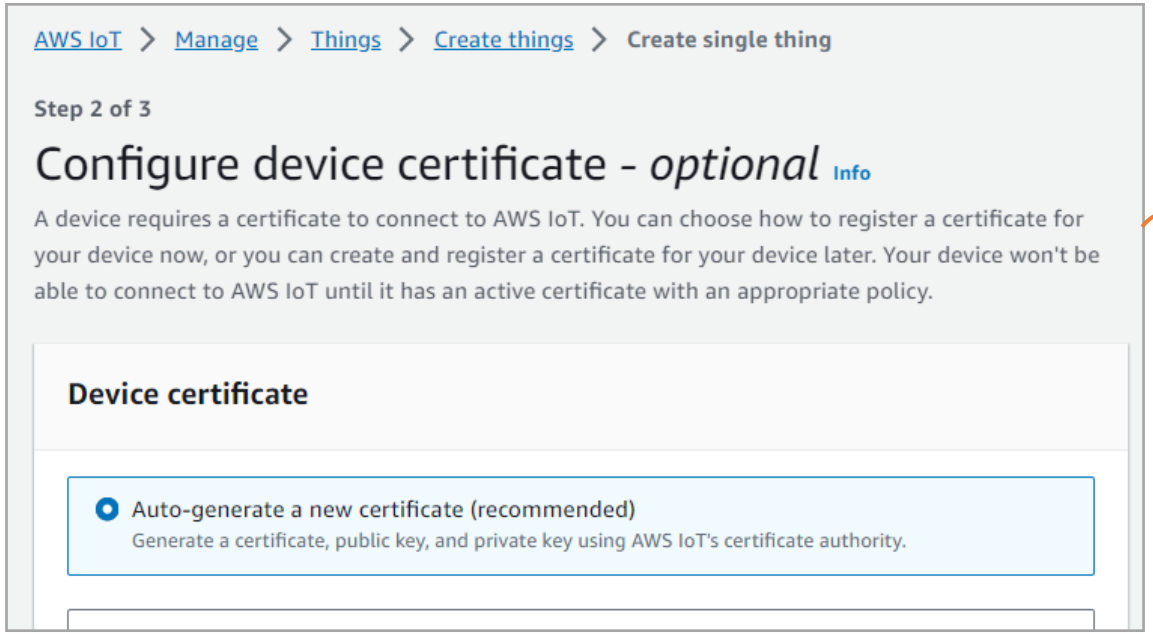
Via AWS IoT Console

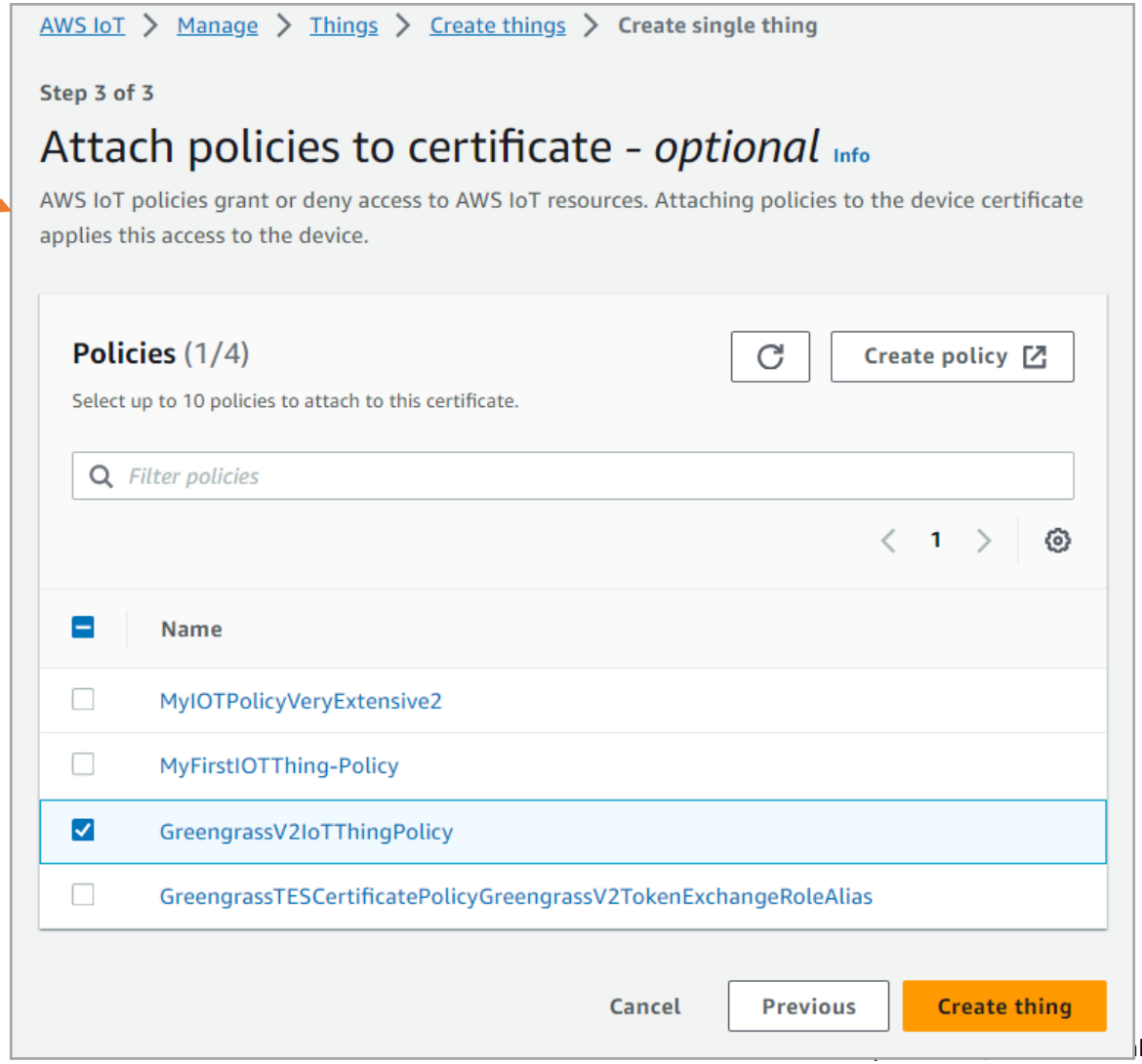




Step 3: Create Client Device (Thing)

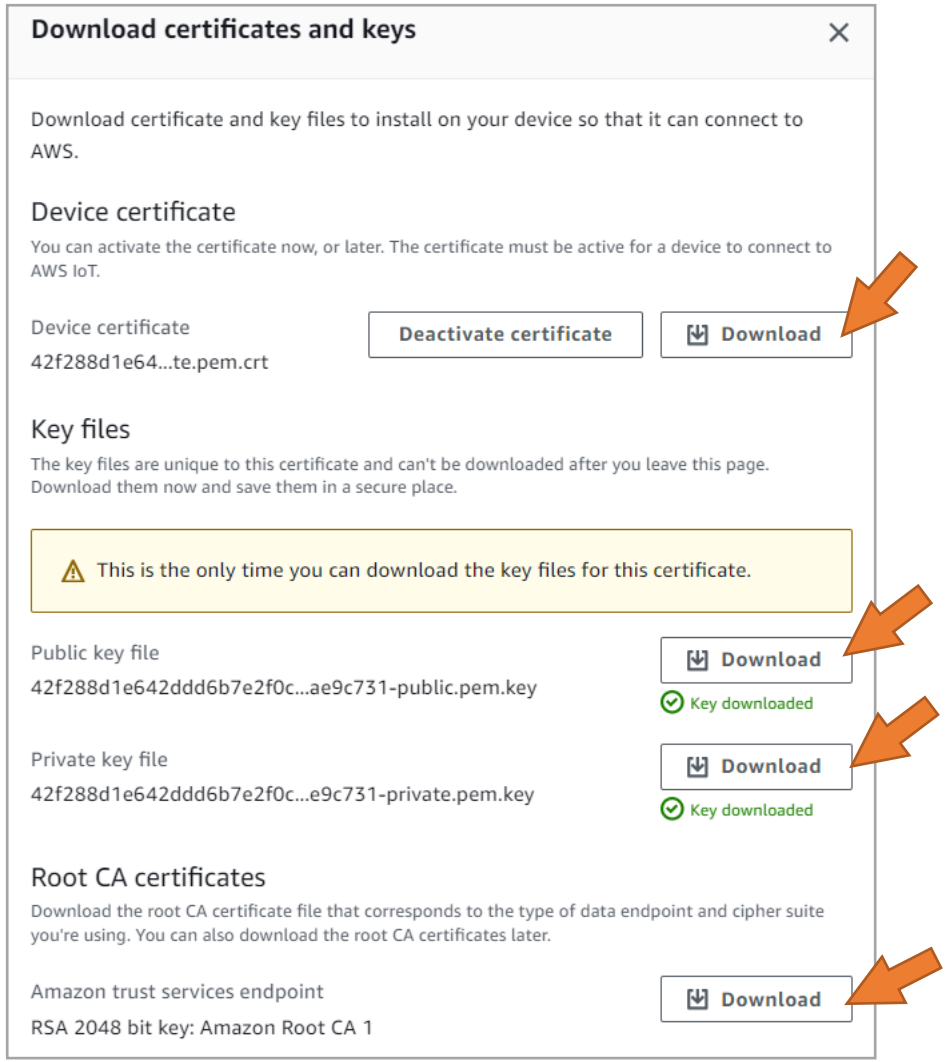
Configure Certificate & Policy





Step 3: Create Client Device (Thing)

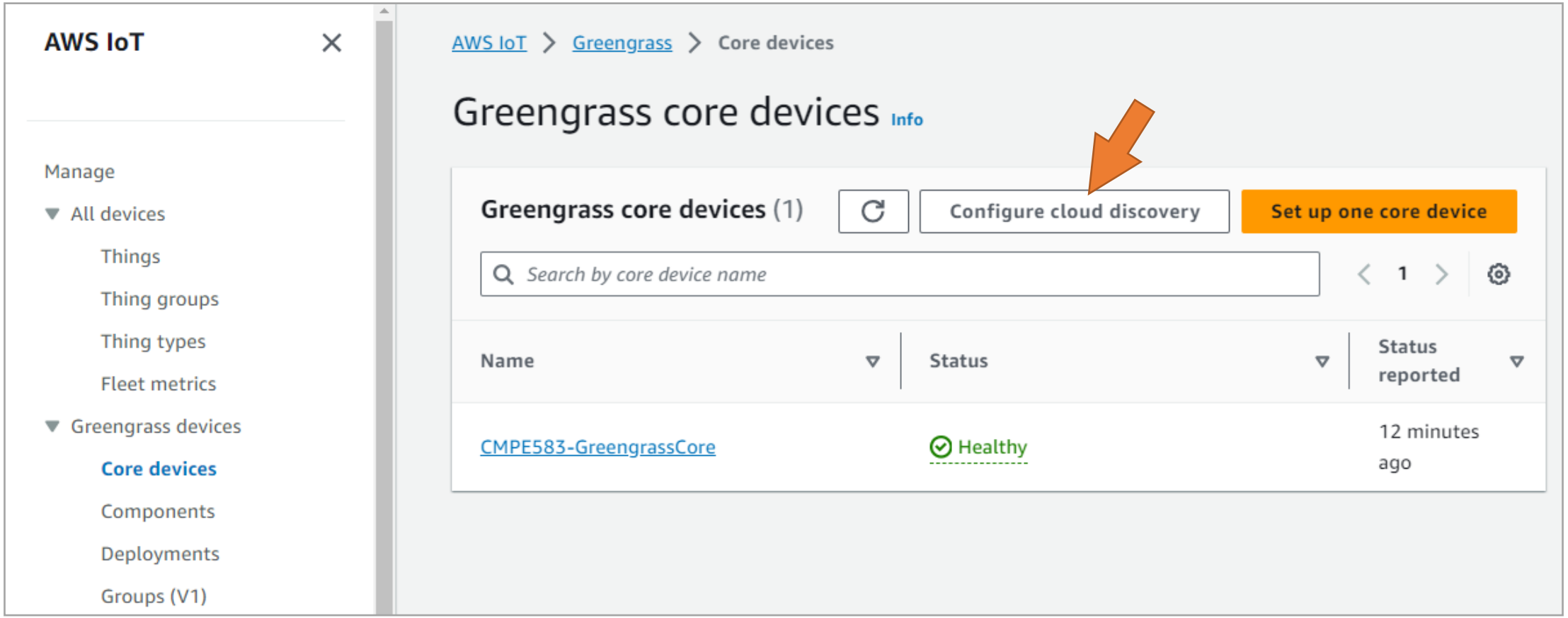
Download Key Files



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

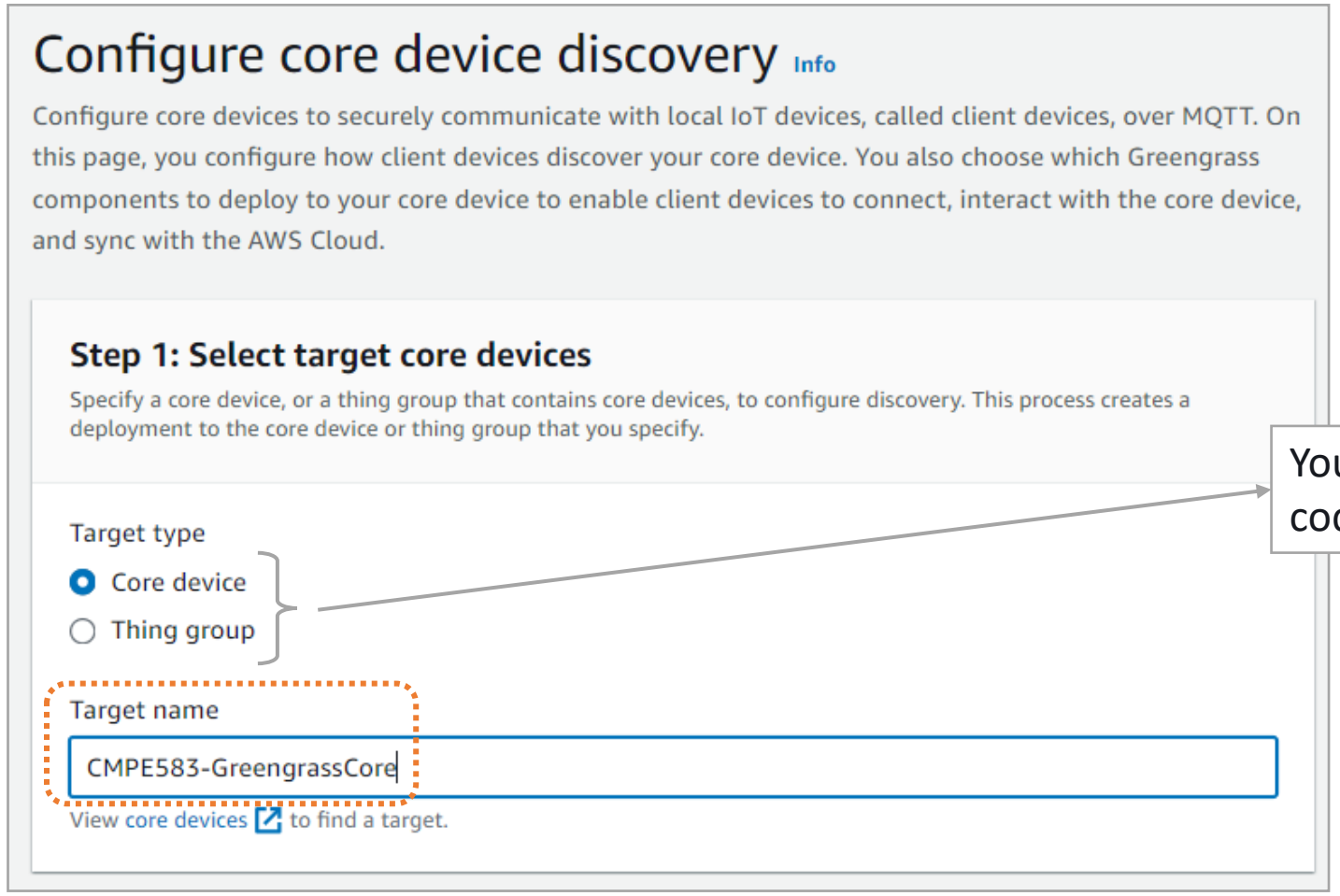
Step 4: Enable Client Device Support

Configure Cloud Discovery



Step 4: Enable Client Device Support

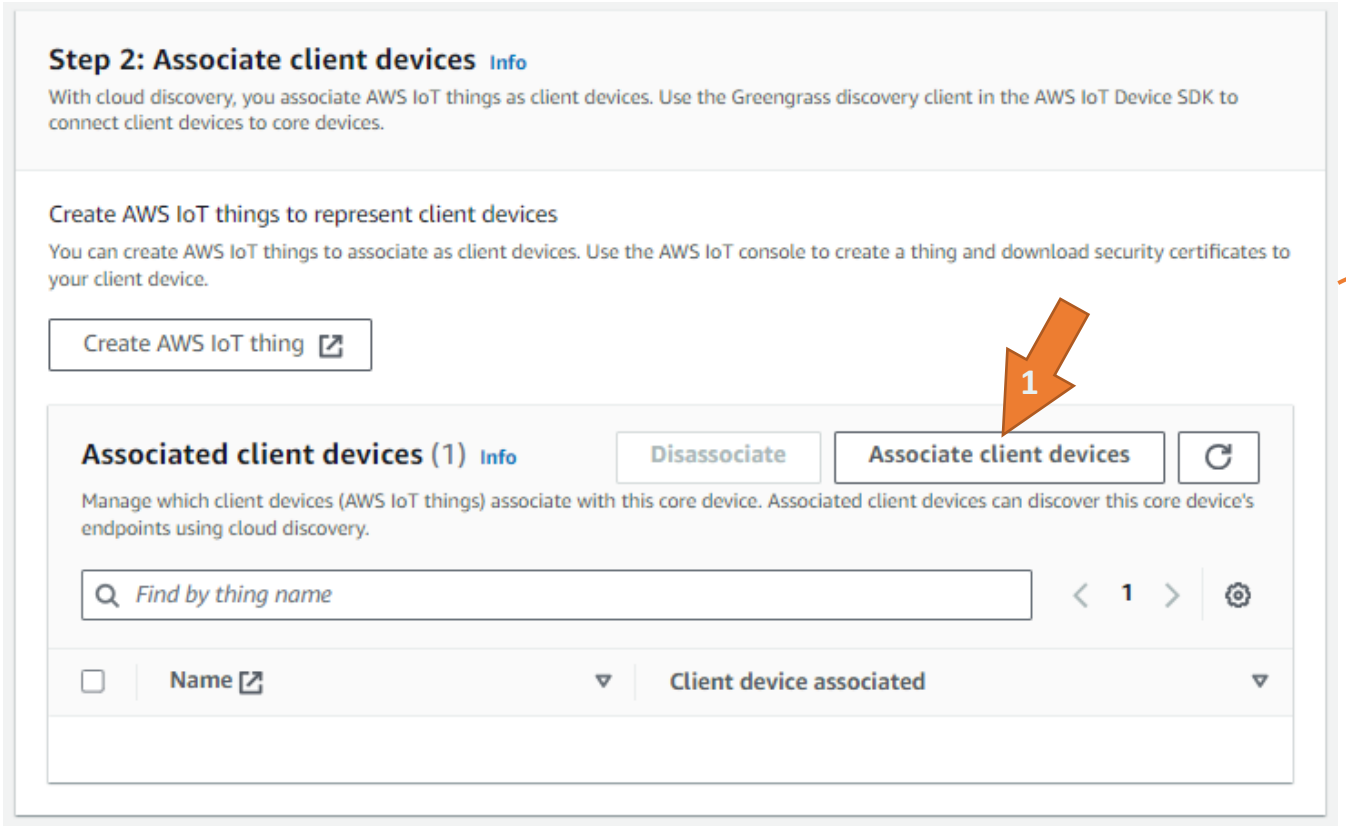
Select Target Core Devices

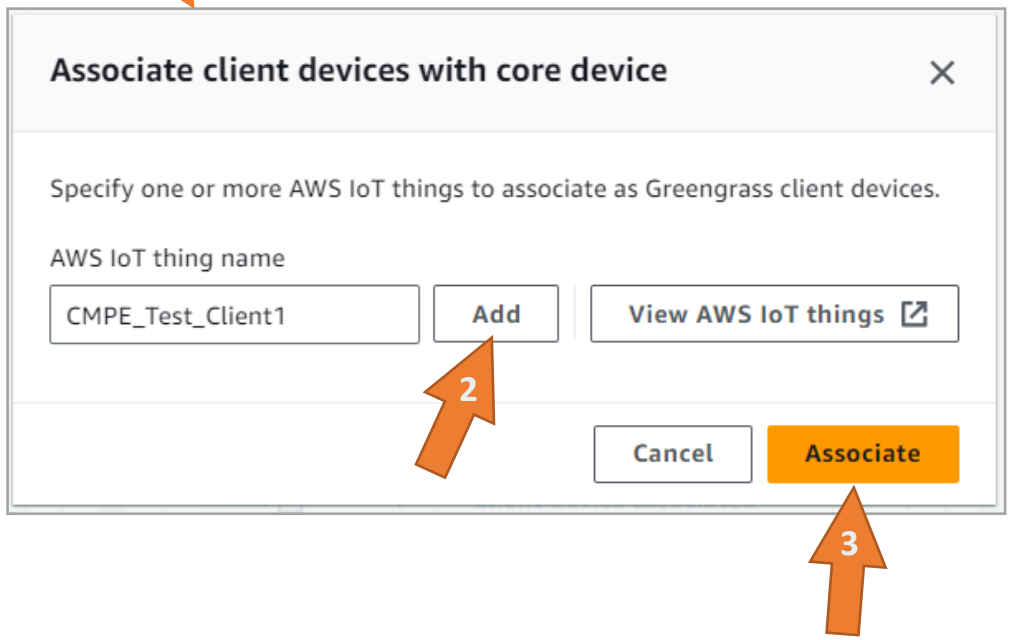


You can configure core device discovery for a code device or a thing group.

Step 4: Enable Client Device Support

Associate Client Devices





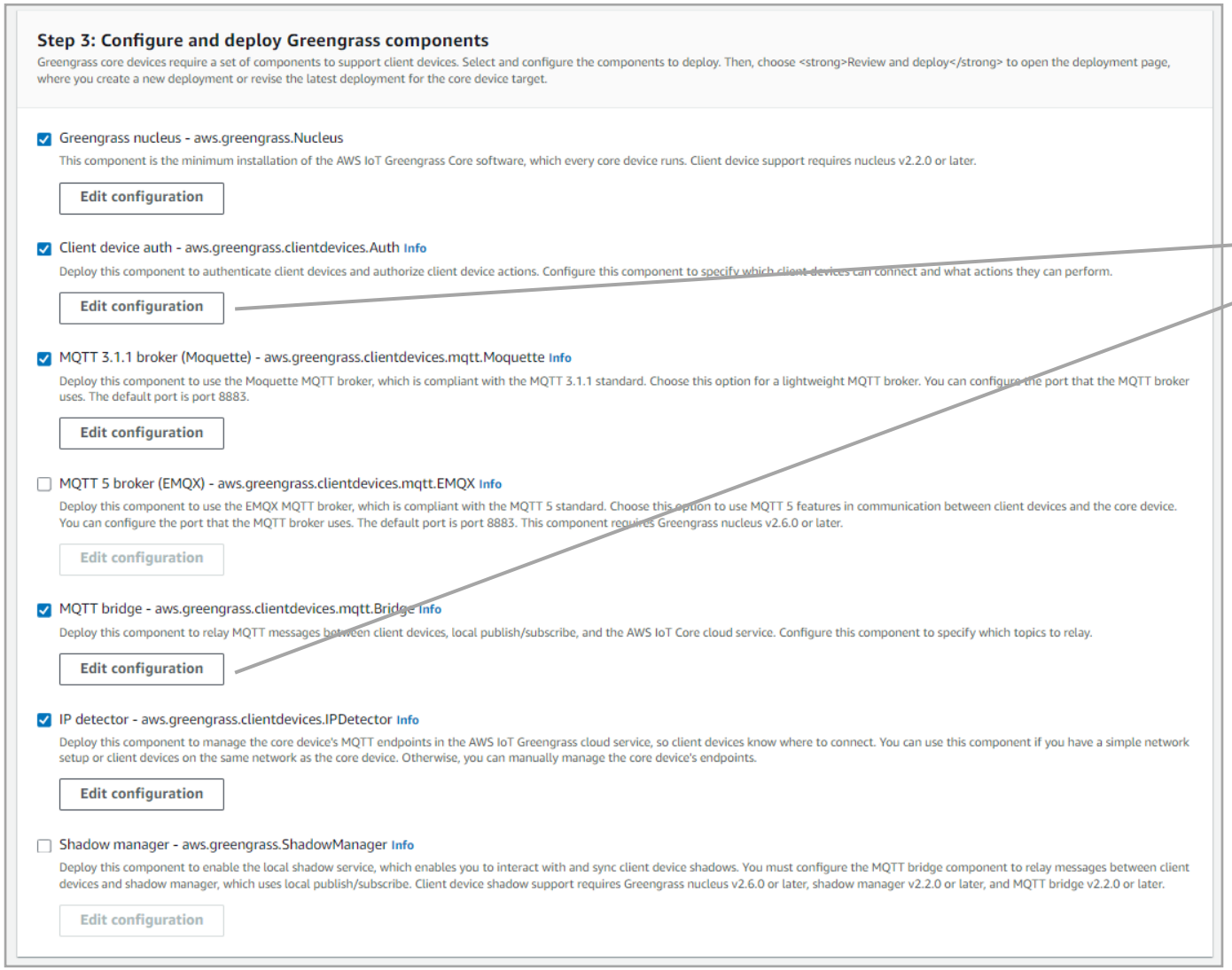
Step 4: Enable Client Device Support

Deploy Required Greengrass Components

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

Step 4: Enable Client Device Support

Configure Required Greengrass Components

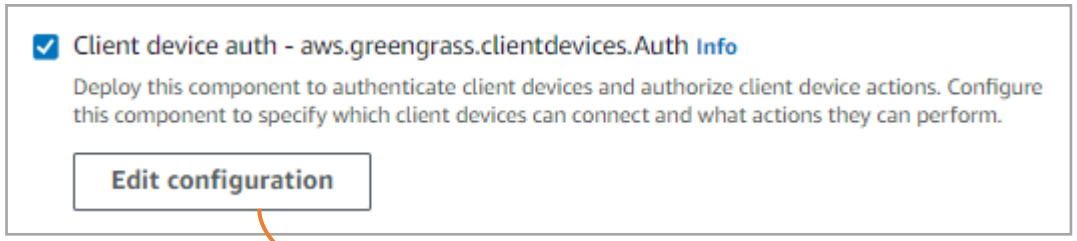


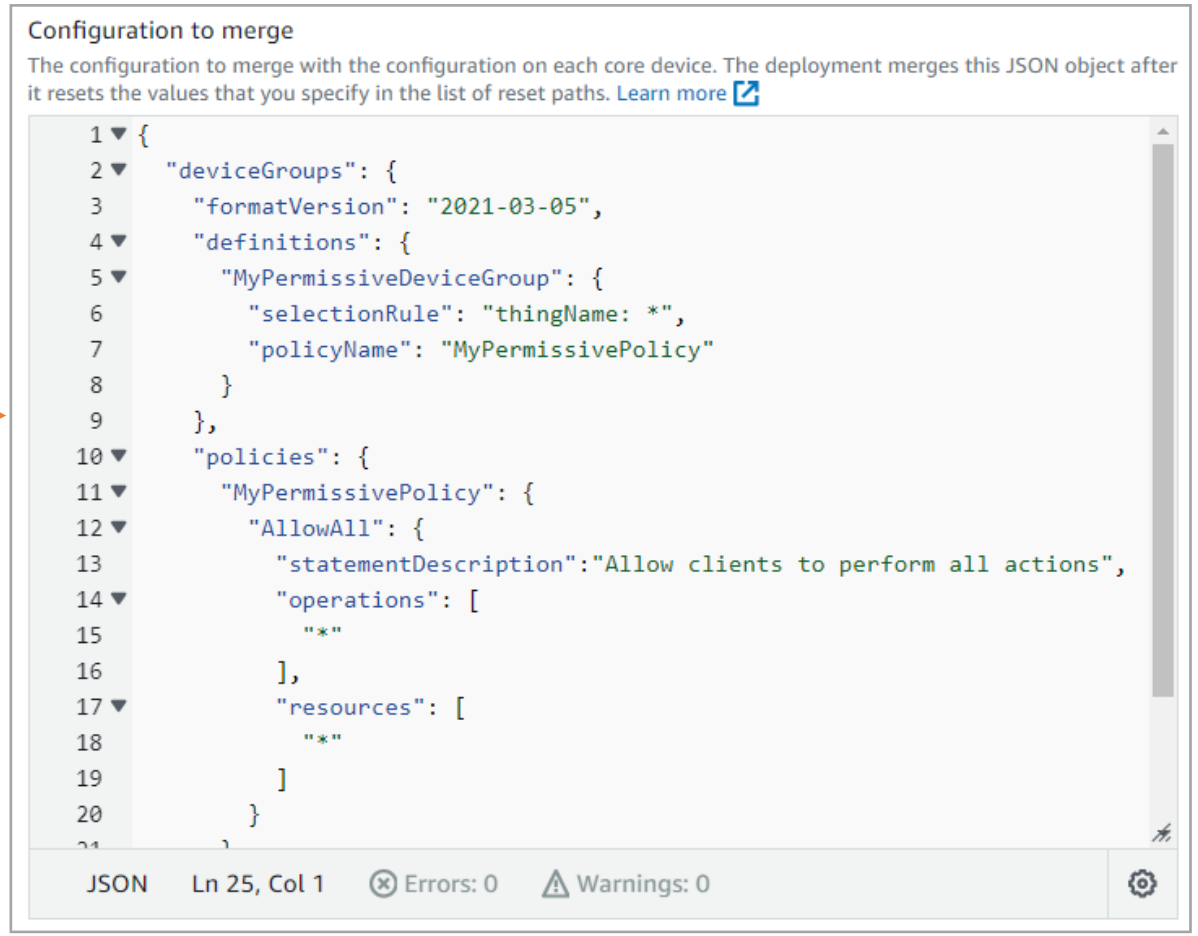
MQTT bridge and client device auth components should be configured before the deployment!

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

Step 4: Enable Client Device Support

Customize aws.greengrass.clientdevices.Auth

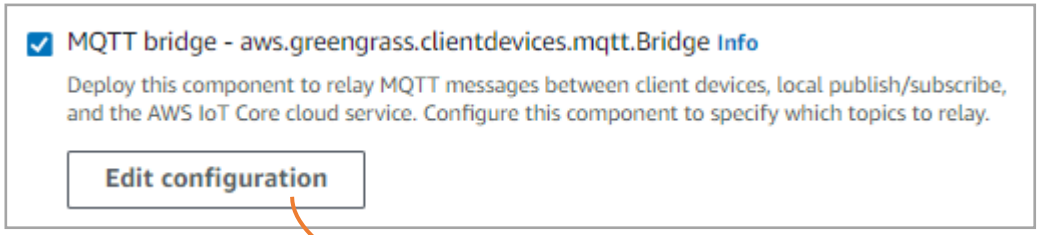


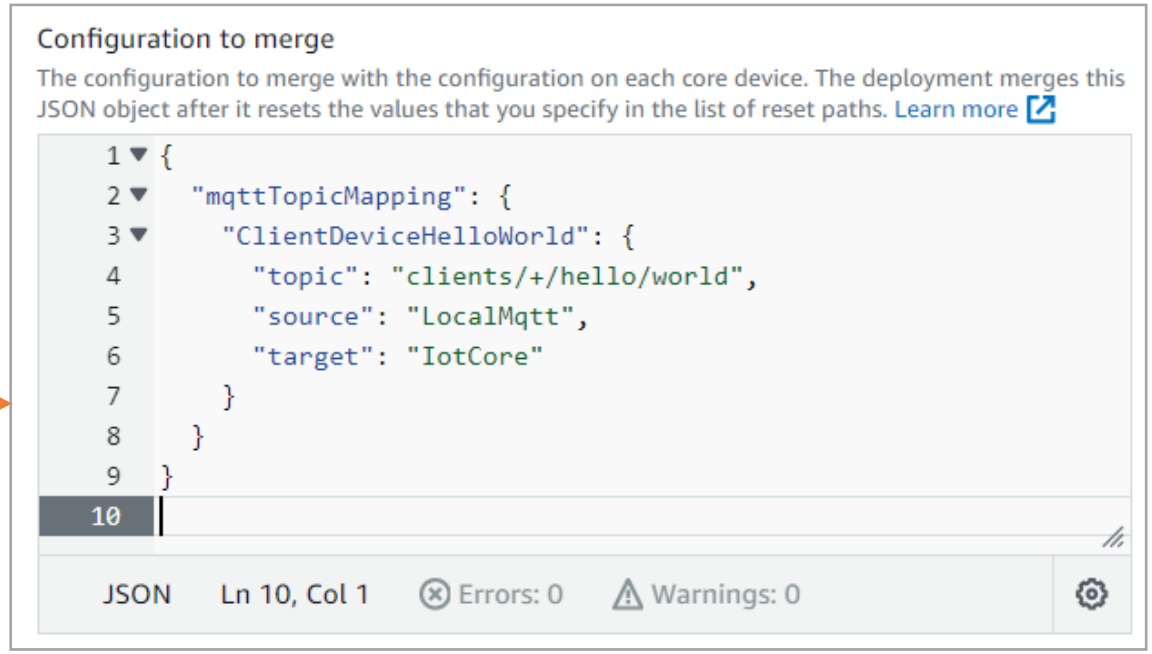


In this example, permissive policy is used to allow all operations. To use a more restrictive policy, check out the policy file in github: ***cmpe583-ClientDeviceAuth.Json***

Step 4: Enable Client Device Support

Customize aws.greengrass.clientdevices.mqtt.Bridge

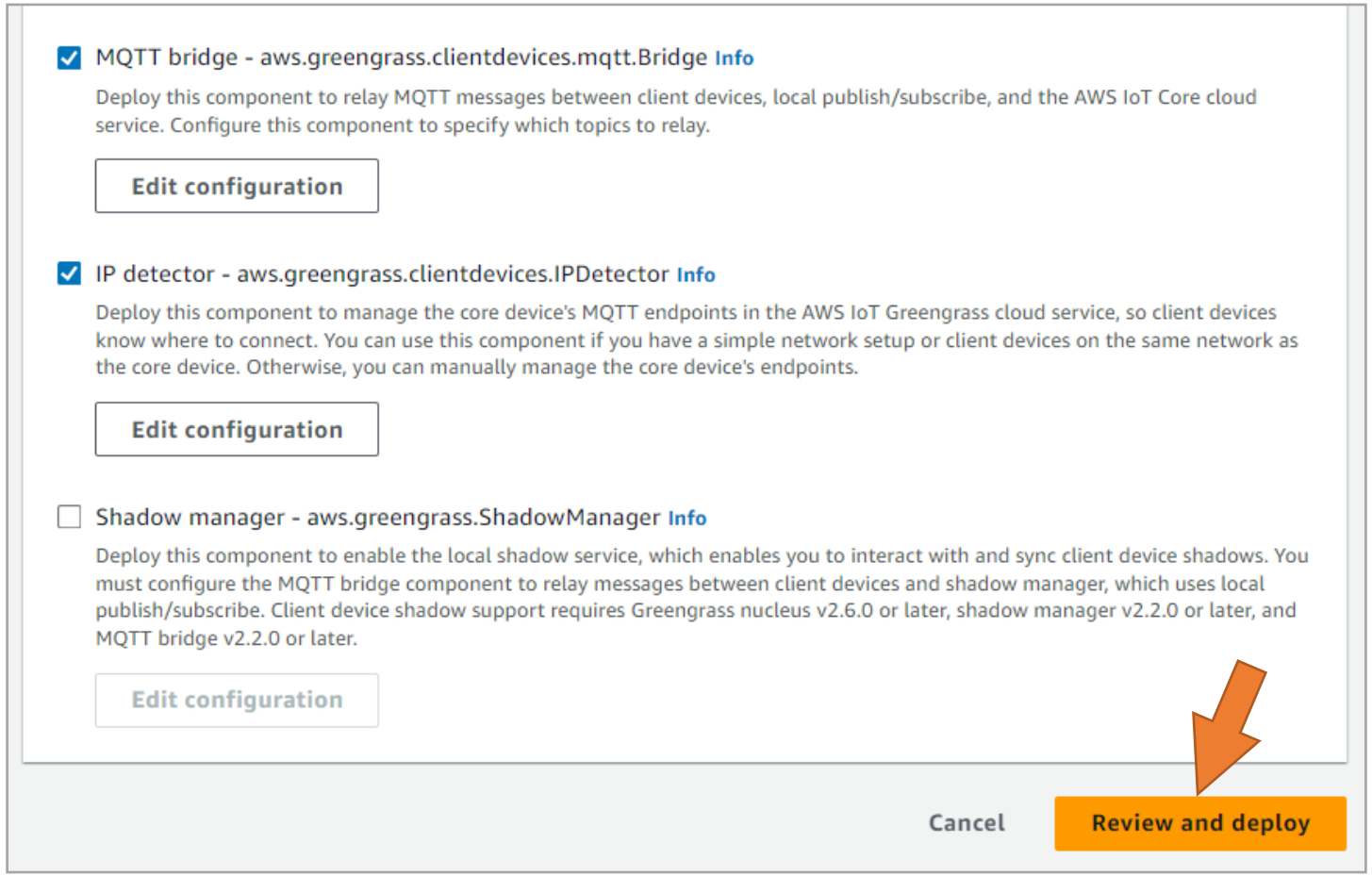




- Relay MQTT messages on the clients/+/hello/world topic filter from client devices to the AWS IoT Core cloud service.
- For example, this topic filter matches the clients/CMPE_Test_Client1/hello/world topic.

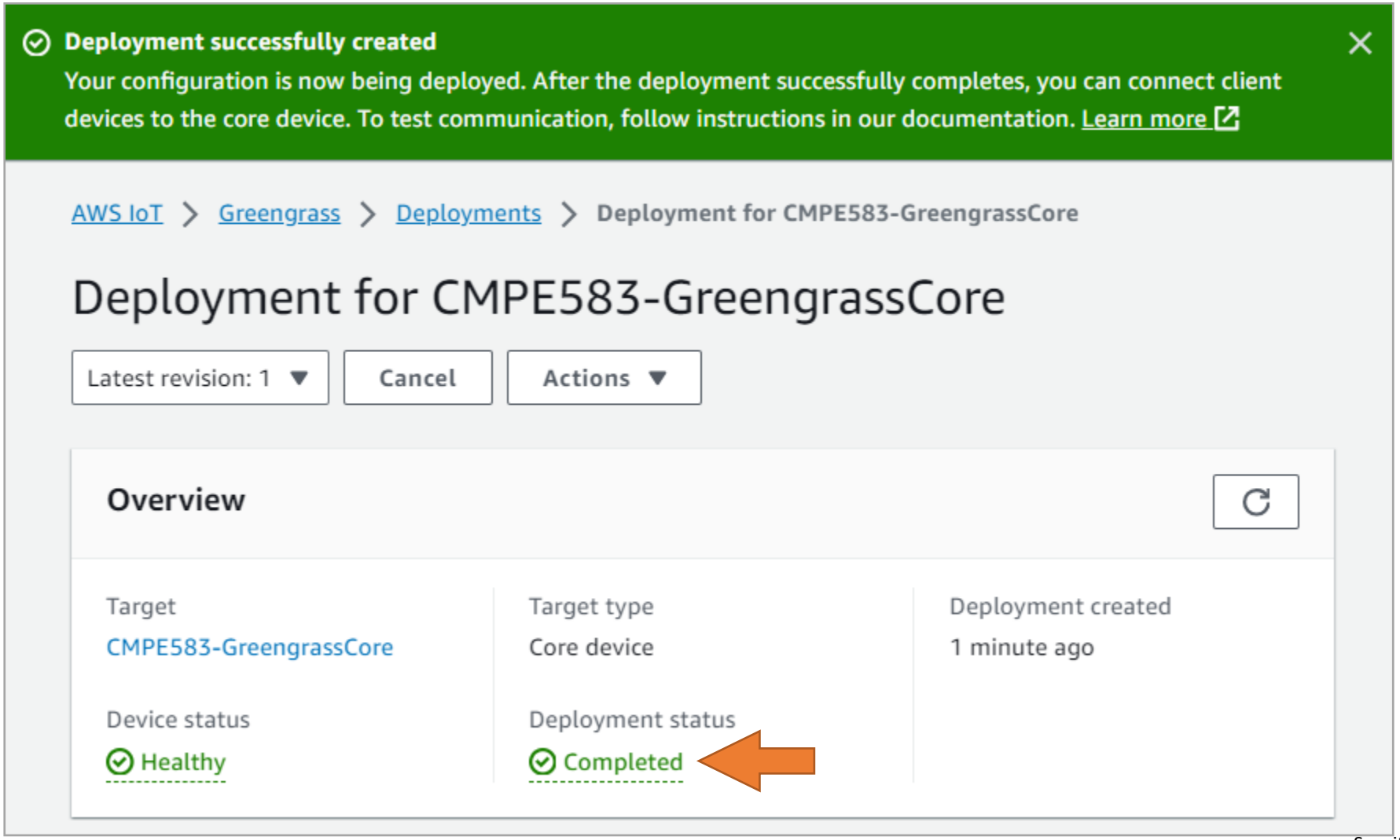
Step 4: Enable Client Device Support

Review & Deploy Greengrass Components



Step 4: Enable Client Device Support

Check Deployment Status



Connect Client Devices

Step 1: Install AWS IoT Device SDKs

Programming Languages

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

Step 1: Install AWS IoT Device SDKs

AWS IoT Device SDK v2 for Python

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

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

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

Install the AWS IoT Device SDK v2 for Python.

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

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

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

Step 2: Run Sample Device Discovery

With samples/basic_discovery.py

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

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

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

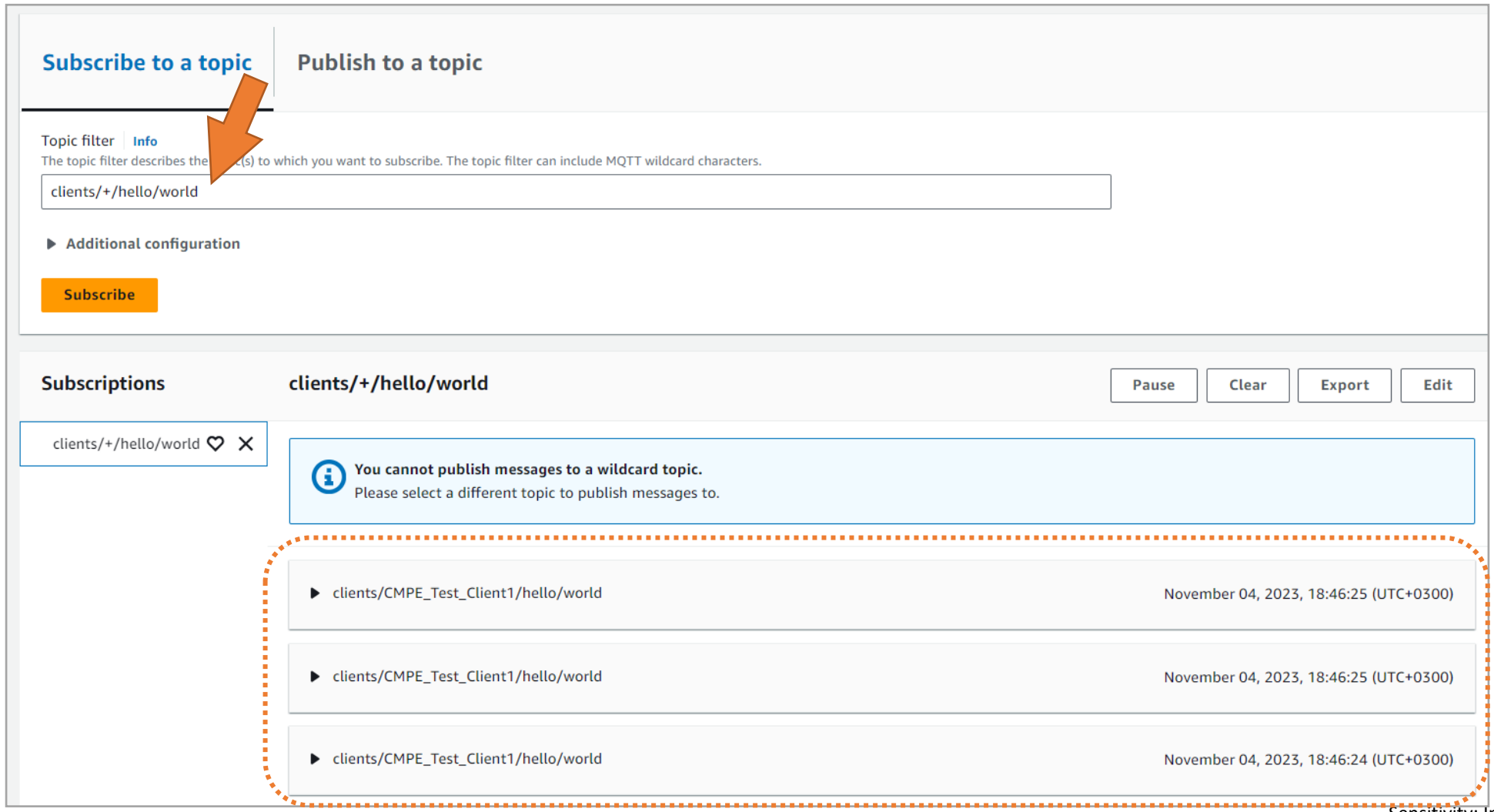
Step 2: Run Sample Device Discovery

Verify Client Device Connection From Terminal

```
cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py     --thing_name CMPE_Test_Client1     --topic
 'clients/CMPE_Test_Client1/hello/world'     --message 'Hello World!'     --ca_file /home/cagatay/shared_folder/AmazonRootCA1.pem     --cert /home/cagatay
/shared_folder/certificate.pem.crt     --key /home/cagatay/shared_folder/private.pem.key     --region eu-central-1     --verbosity Warn
Performing greengrass discovery...
[WARN] [2023-11-04T15:46:21Z] [00007f5a87fff640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice-CMPE583-GreengrassCore',
cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_
discovery.ConnectivityInfo(id='10.0.2.15', host_address='10.0.2.15', metadata='', port=8883)])], certificate_authorities=['-----BEGIN CERTIFICATE-----\nMI
ID1TCCAr2gAwIBAgIVAMDxNyNBeW2Jja8e093NSaPTIV8WMA0GCSqGSIb3DQEB\nCwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBJbmMuMRww\nGgYDVQQLDBNBbWF6b24gV2Vi
IFNlcnZpY2VzMRMwEQYDVQQIDApXYXNoaW5ndG9u\nMRAwDgYDVQQHDAdTZWF0dGxlMRswGQYDVQQDDBJHcmVlbmdyYXNzIENvcmUgQ0Ew\nHhcNMjMxMTA0MTMwMTM5WhcNMjgxMTAyMTMwMTM5WjCBiT
ELMAkGA1UEBhMCVVMx\nGDAWBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMBoGA1UECwwTQW1hem9uIFdlYiBT\nZXJ2aWNlczETMBEGA1UECAwKV2FzaGluZ3RvbjEQMA4GA1UEBwwHU2VhdHRsZTEb\nMB
kGA1UEAwwSR3JlZW5ncmFzcyBDb3JlIENBMIIBIjANBgkqhkiG9w0BAQEFAAOC\nAQ8AMIIBCgKCAQEAmZ2lfIUSehjcUmSgkm1uCt8VoKcQtWIoy0pzVmyNaR2RnDFr\n48huzRAT+Ee6dVxyLMRcsw1n
cJM62cPv8wATNDgC/IMdtotLLqwAGKdcFfFiWvtx\nEZ1zfN8JsQJ8Mutr4X+PfFmFSqxSTdfTXB2l7Z0SX0SE7auaLzHcH8HTDYAklp2h\nXeK4VMxiKbHne4KdoPbQgk8m83BniAOWOKOTab9wFjwi+1
wd875jV3r1SS64uz+9\ncJ6Pp+4Kv+cvjZyMWPTnSKI4QoXlbQd+92lXZwqtGCTvzAPdLm5E/B8ber2/lb2s\nwJ4f8A7jeQs5VtDZmVoeXLd8Sdm/ocji4+pGqQIDAQABozIwMDAPBgNVHRMBAf8E\nBT
ADAQH/MB0GA1UdDgQWBBQyiqUipUYOXyyDknCF/imtYlYyLzANBgkqhkiG9w0B\nAQsFAAOCAQEAYfoiiBX5fNJRpm/MW1VVdE62f7K+3OatTPoroZa6CTgULZhdTnWG\nWP5MIZt9iKq3AQzgma7aQEt/
dKJJL3nmDqsY/fHRjjjcF6SSu1W4Fp32BFp3mi57\naeICvFqMgHztyBB5WJVsU7EktWSn0PjWSgcBWticXFw8aOG55HS5rrUqi9mBYWSm\njo/7rVsX/UOkUHP19sBu6JrJarvibkUdyxY66mOEpLMmf9
My/11TIL5s2rhpqRSX\nR9DsiyGAq7KXaJzMsxWn1DxvlyRfjNo3R3W0ICWq1F6y6RtwU+ofAoTxMbC25XQ0\ncYI/qnKkj0/K4ycY0QPZN7qr76I5INP5iA==\n-----END CERTIFICATE-----\n'])
])
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/hello/world: {"message": "Hello World!", "sequence": 0}

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

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

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

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

Step 3: Verify Client Device Connection (MQTT Bridge)

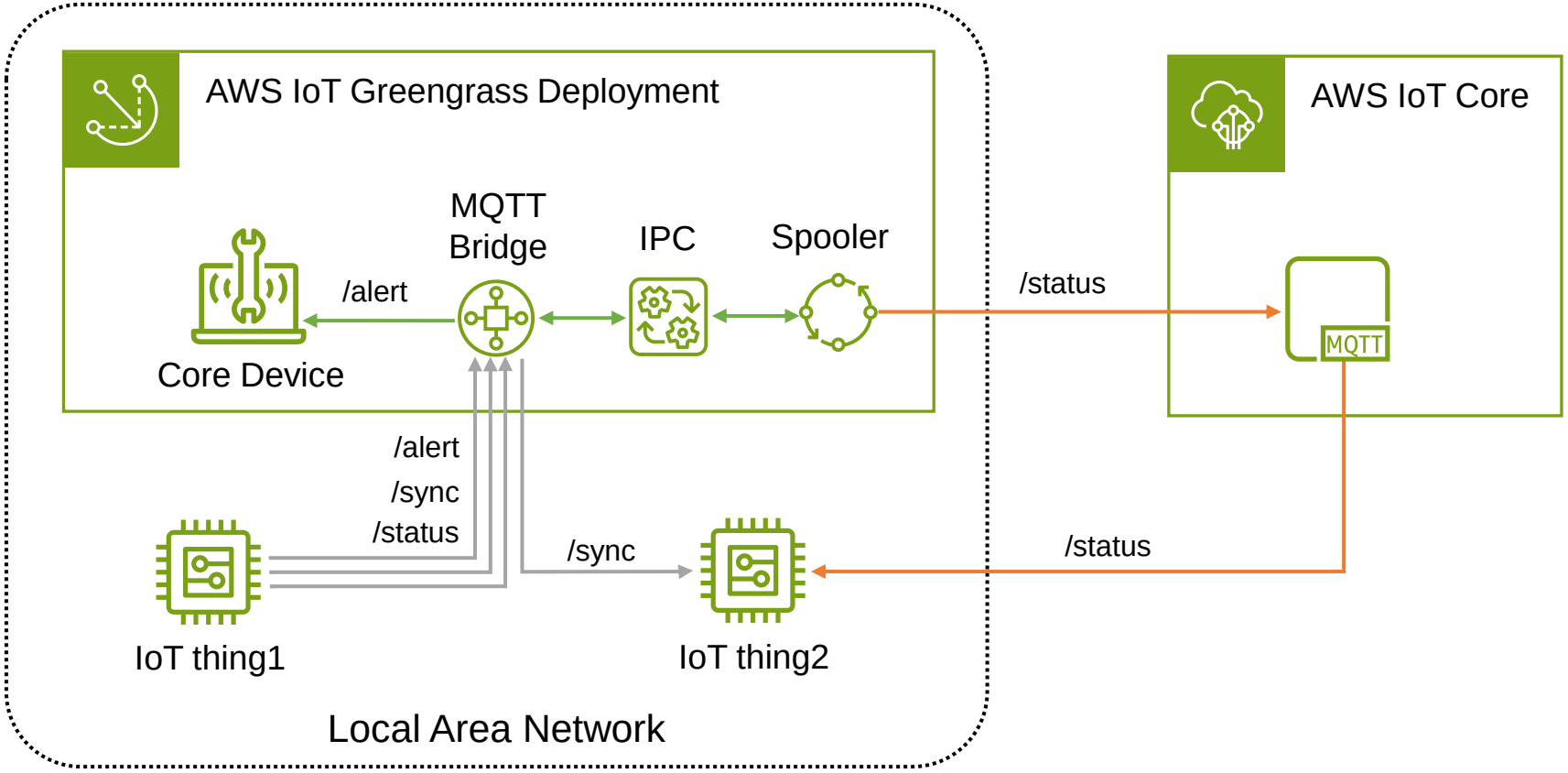
Verify From MQTT Test Client



Communicate with Client Devices

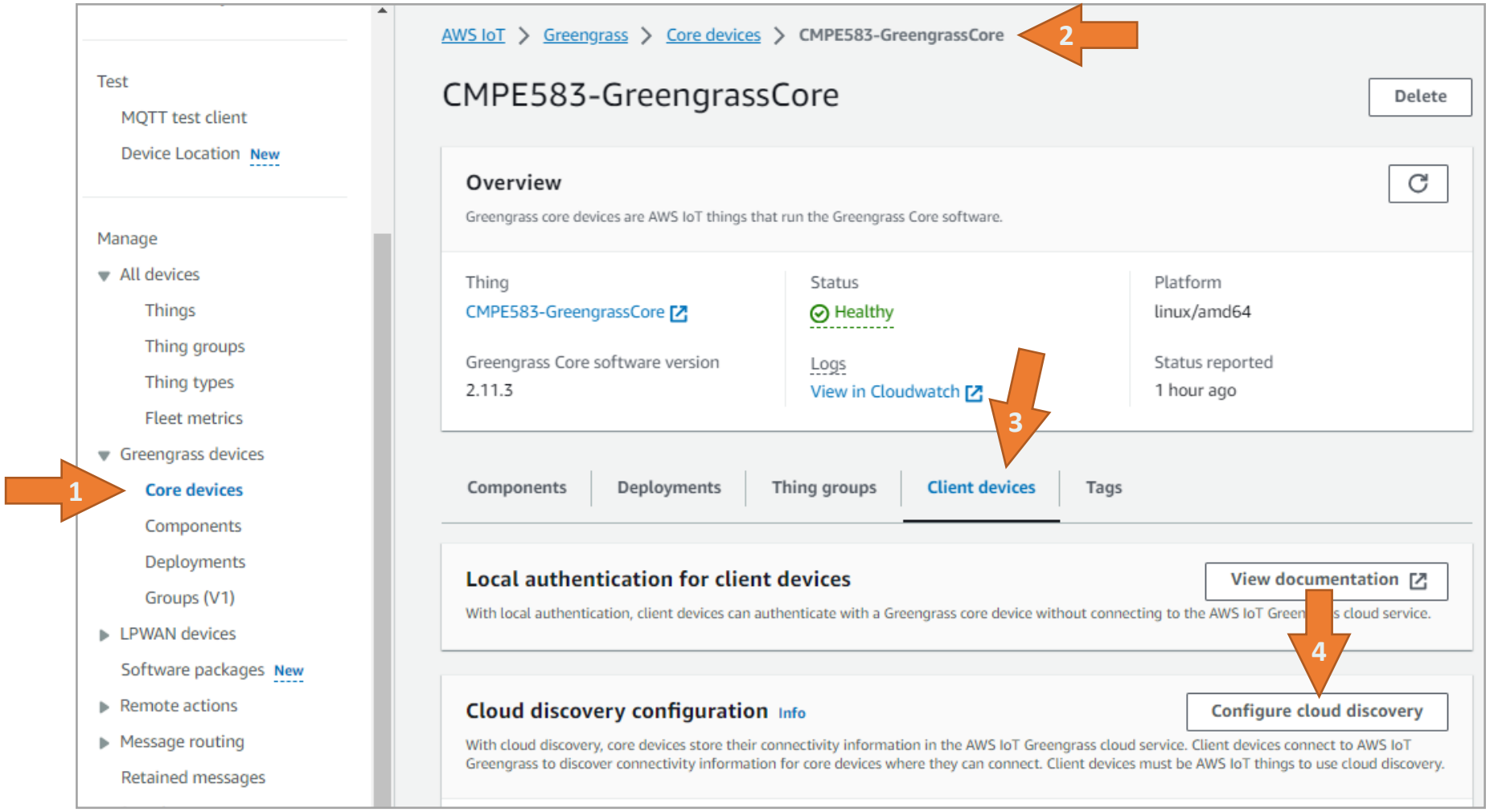
Communicate with Client Devices

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



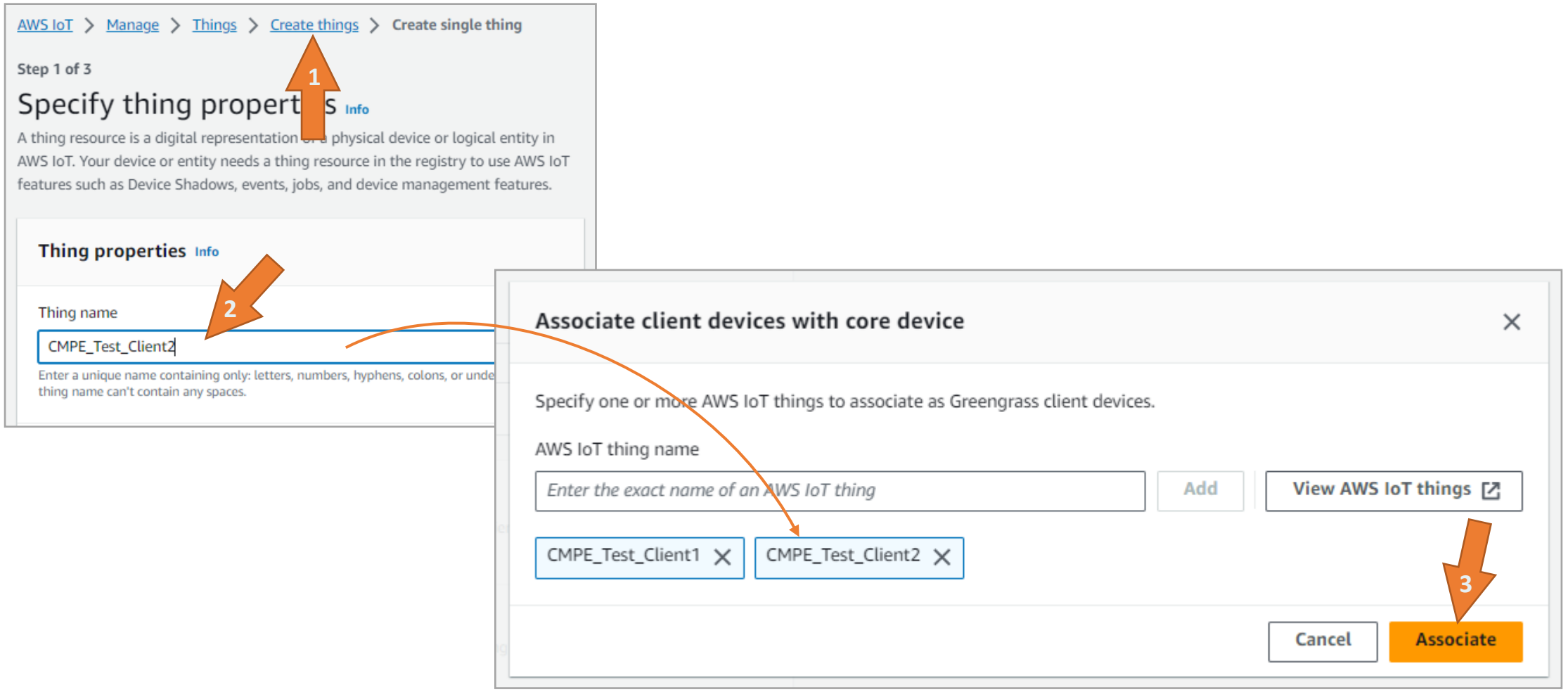
Step 1: Reconfigure Cloud Discovery

To Edit the MQTT Bridge



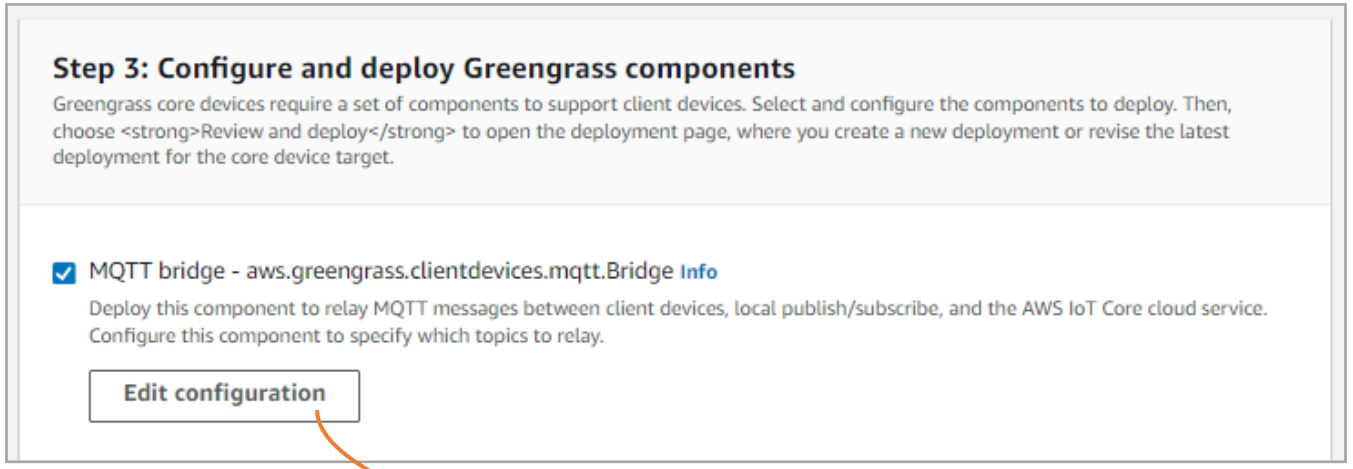
Step 1: Reconfigure Cloud Discovery

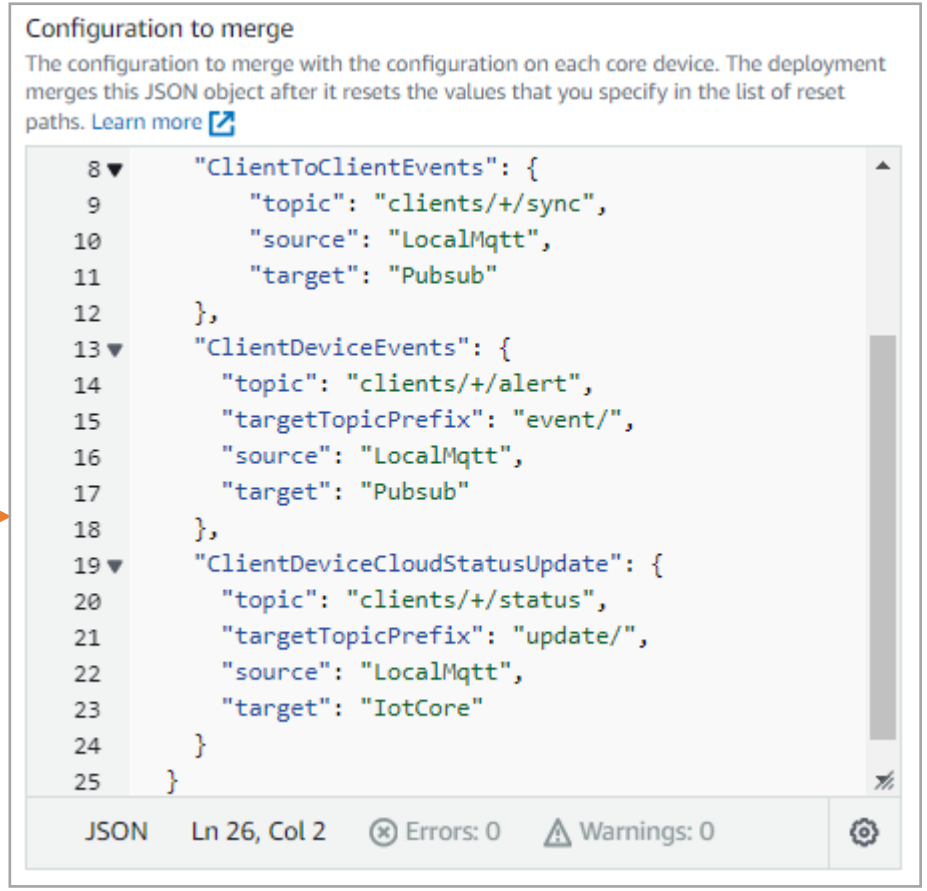
Create Another Client and Associate It With Core Device



Step 1: Reconfigure Cloud Discovery

Reconfigure MQTT Bridge





Step 1: Reconfigure Cloud Discovery

New MQTT Topic Mapping

Configuration to merge

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

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

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

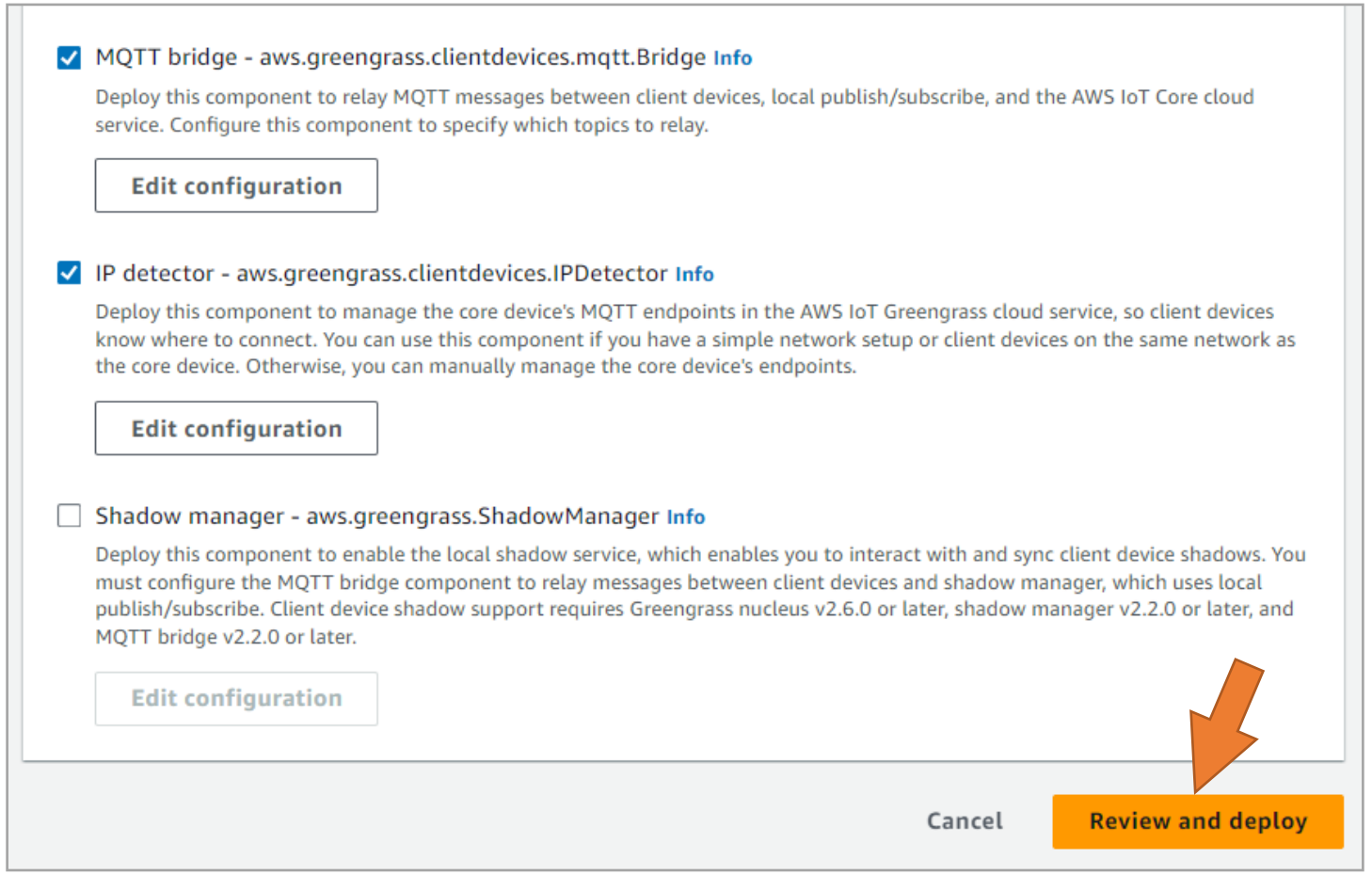
Relay messages from client devices to other clients on topics that match the clients/+/sync topic filter through MQTT Bridge on Core Device.

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

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

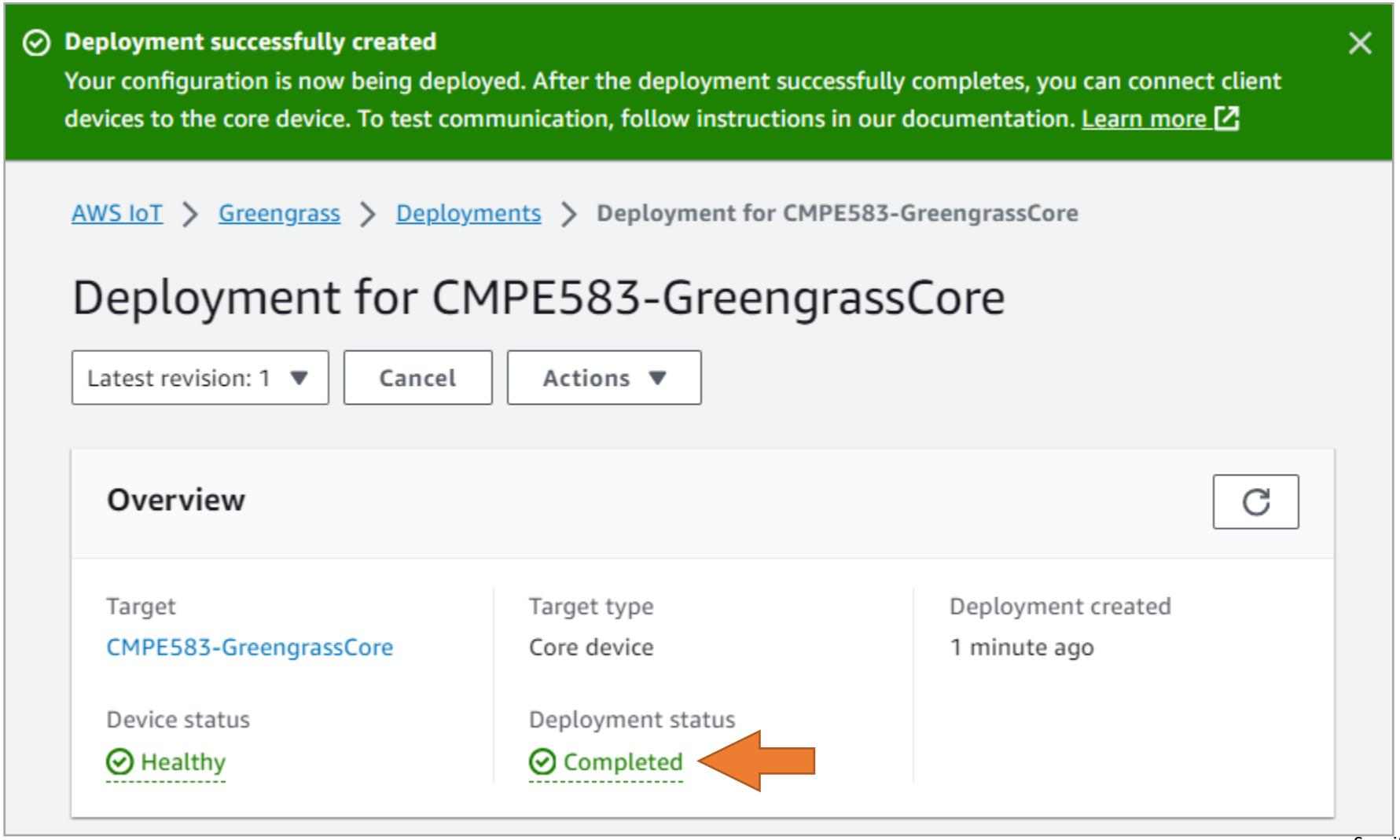
Step 1: Reconfigure Cloud Discovery

Review & Deploy the New Configuration



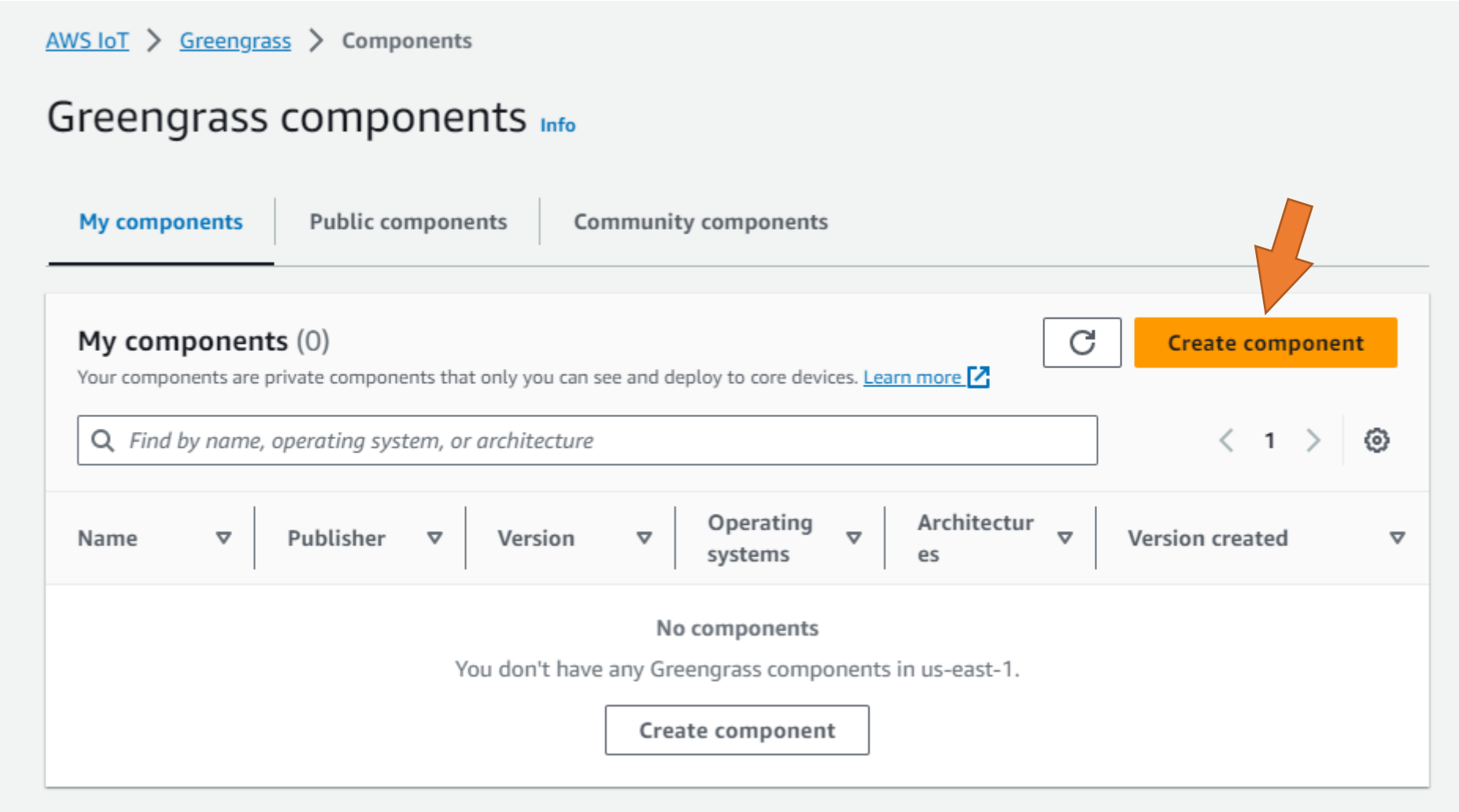
Step 1: Reconfigure Cloud Discovery

Check Deployment Status



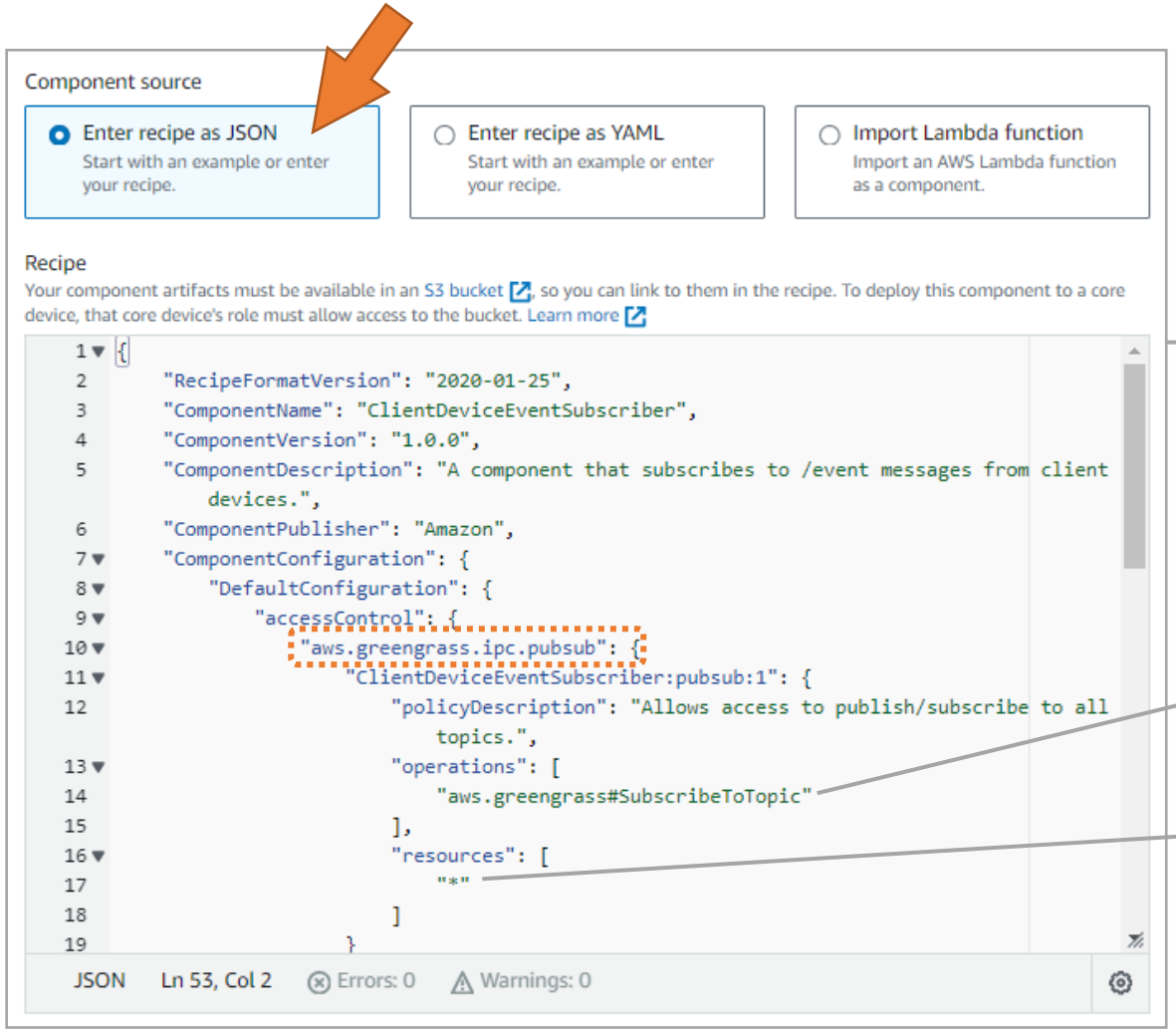
Step 2: Create a Custom Component

To Suscribe Local MQTT Events



Step 2: Create a Custom Component

Configure accessControl for Local Messages



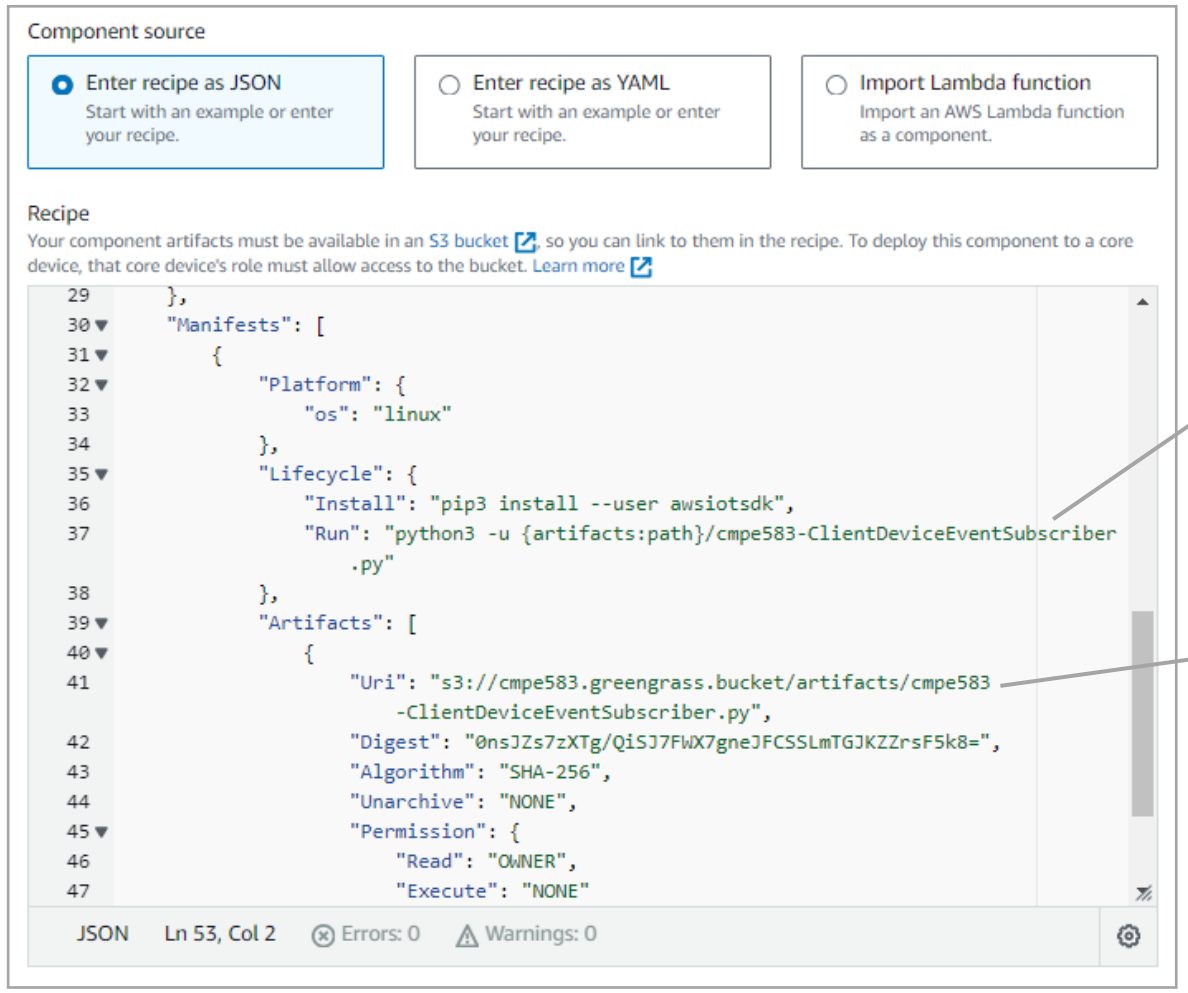
Check GitHub to see whole contents in ***cmpe583-ClientDeviceEventSubscriber.json***

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

Allow access to all topics.

Step 2: Create a Custom Component

Configure Artifacts in the Recipe

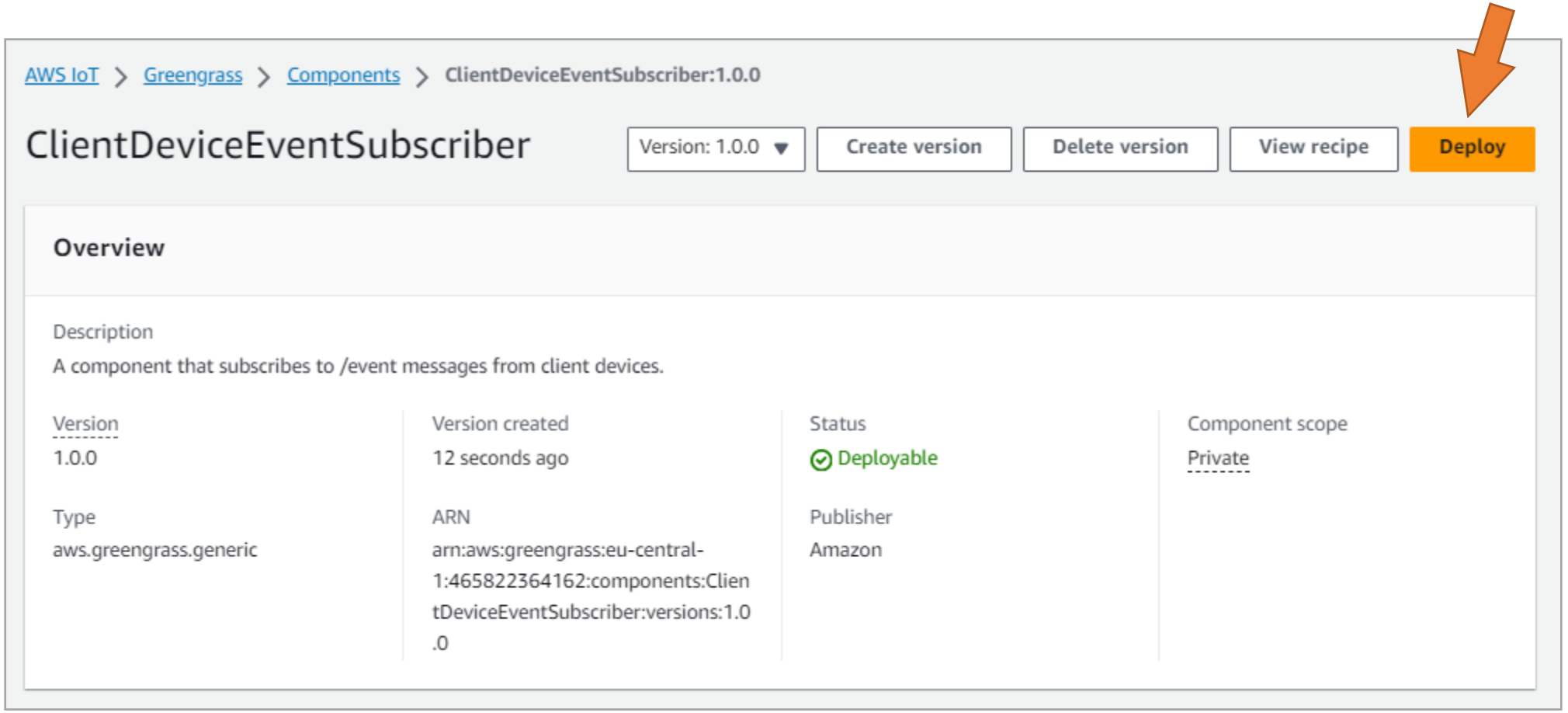


Check GitHub to see source code in python file: ***cmpe583-ClientDeviceEventSubscriber.py***

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

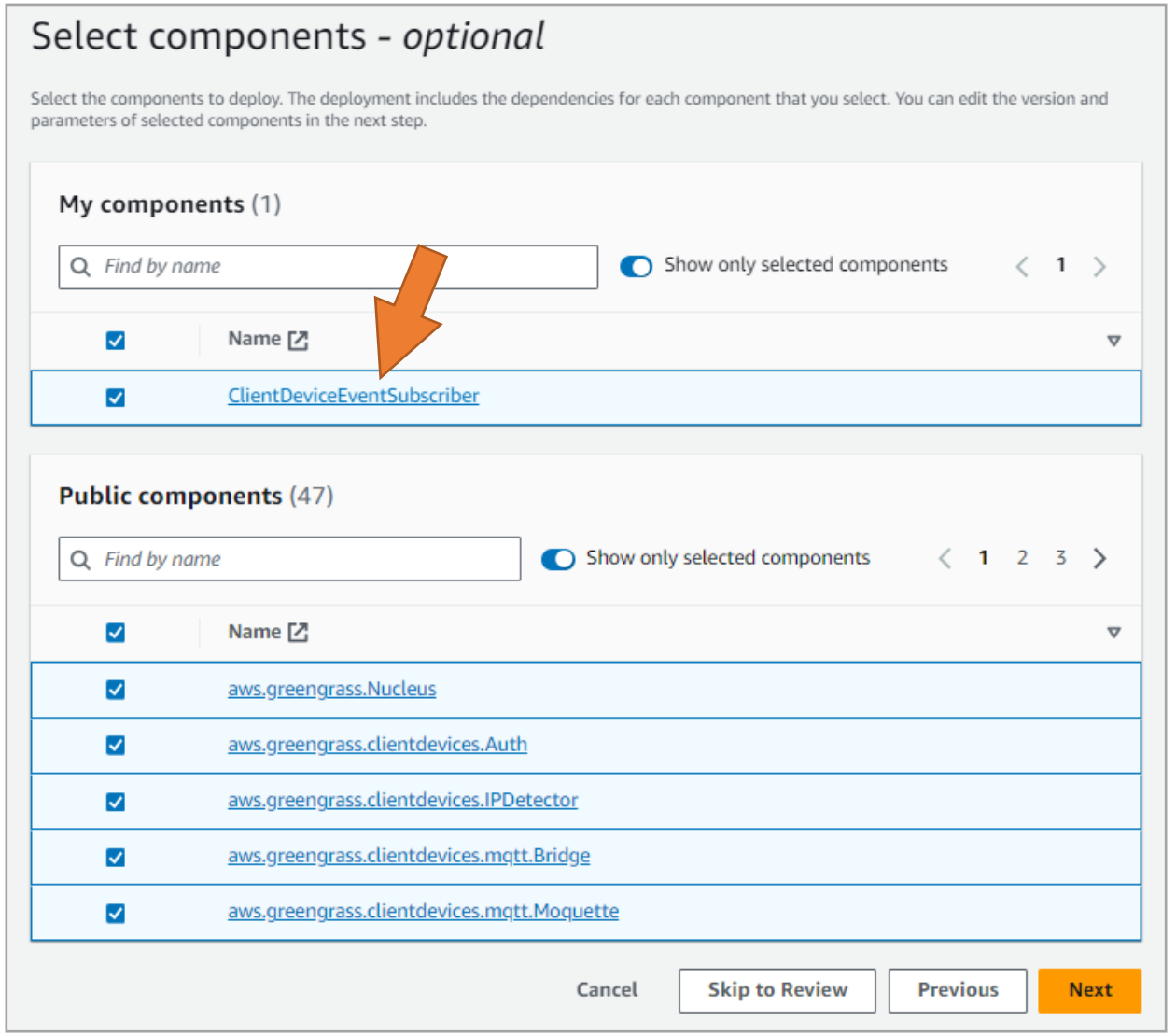
Step 3: Deploy Your Component

Via AWS IoT Greengrass (Console)



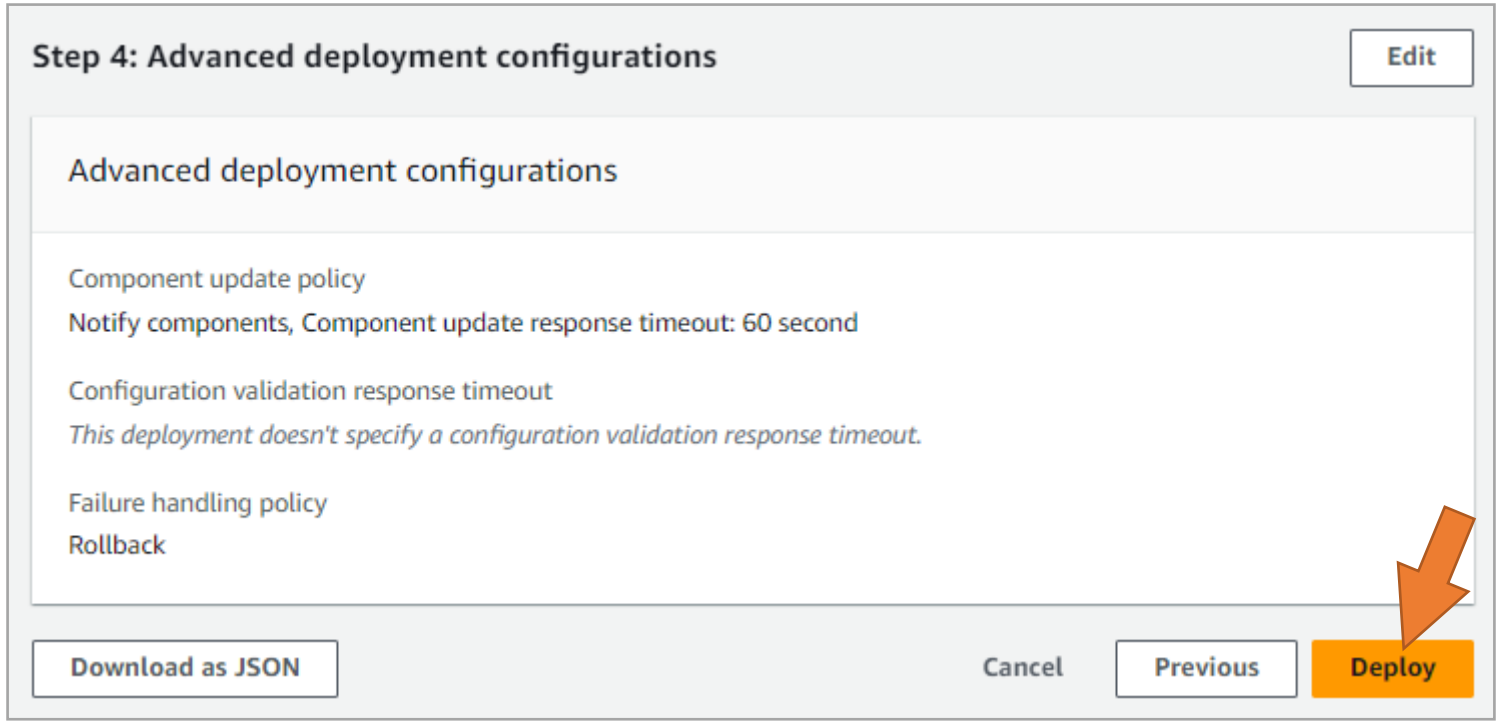
Step 3: Deploy Your Component

Select the Component



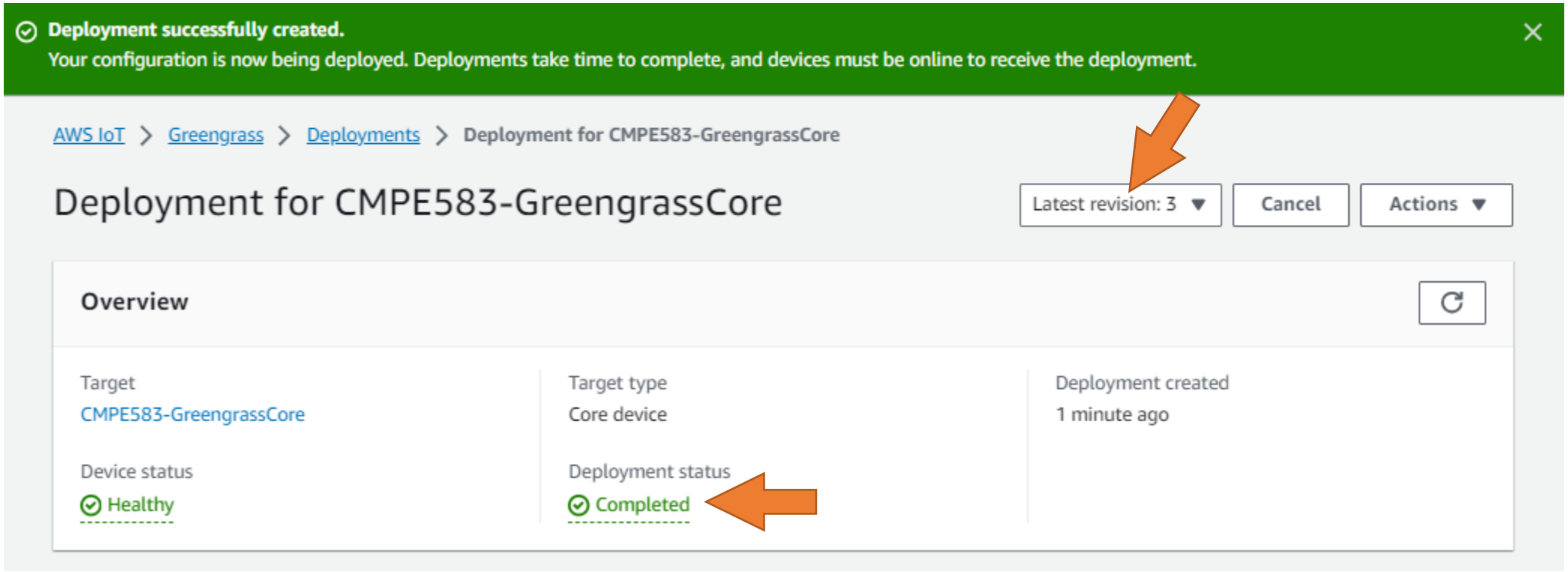
Step 3: Deploy Your Component

Review & Deploy



Step 3: Deploy Your Component

Check Deployment Status on AWS IoT Console



Step 3: Deploy Your Component

Check Deployment Status on Core Device

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

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

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

Publish Topic to Trigger Subscribers (Core Device)

- Publish the topic with basic_discovery.py script.

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

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

This message will be relayed to Core Device

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

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

Verify Process on Terminal

```
cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
    --thing_name CMPE_Test_Client1 \
    --topic 'clients/CMPE_Test_Client1/alert' \
    --message 'Production Anomaly Detected. Check the Production Plant!' \
    --ca_file /home/cagatay/shared_folder/AmazonRootCA1.pem \
    --cert /home/cagatay/shared_folder/certificate.pem.crt \
    --key /home/cagatay/shared_folder/private.pem.key \
    --region eu-central-1 \
    --verbosity Warn \
   --max_pub_ops 1
Performing greengrass discovery...
[WARN] [2023-11-05T08:38:23Z] [00007f368d0fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice-CMPE583
-GreengrassCore', cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCor
e', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address='10.0.2.15', metadata='', port=8883)])], cer
tificate_authorities=['-----BEGIN CERTIFICATE-----\nMIID1TCCAr2gAwIBAgIVAMDxNyNBeW2Jja8e093NSaPTIV8WMA0GCSqGSIb3DQEB\nCwUAMIGJMQswCQYDVQ
QGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBJbmMuMRww\nGgYDVQQLDBNBbWF6b24gV2ViIFNlcnZpY2VzMRMwEQYDVQQIDApXYXNoaW5ndG9u\nMRAwDgYDVQQHDAdTZWF0dG
xlMRswGQYDVQQDDBJHcmVlbmdyYXNzIENvcmUgQ0Ew\nHhcNMjMxMTA0MTMwMTM5WhcNMjgxMTAyMTMwMTM5WjCBiTELMAkGA1UEBhMCVVMx\nGDAWBgNVBAoMD0FtYXpvbi5jb2
0gSW5jLjEcMBoGA1UECwwTQW1hem9uIFdlYiBT\nZXJ2aWNlczETMBEGA1UECAwKV2FzaGluZ3RvbjEQMA4GA1UEBwwHU2VhdHRsZTEb\nMBkGA1UEAwwSR3JlZW5ncmFzcyBDb3
JlIENBMIIBIjANBgkqhkiG9w0BAQEFAAOC\nAQ8AMIIBCgKCAQEAmZ2lfIUSehjcUmSgkm1uCt8VoKcQtWIoy0pzVmyNaR2RnDFr\n48huzRAT+Ee6dVxyLMRcsw1ncJM62cPv8w
ATNDgC/IMdtotLLqwAGKdcFfFiWvtx\nEZ1zfN8JsQJ8Mutr4X+PfFmFSqxSTdfTXB2l7Z0SX0SE7auaLzHcH8HTDYAklp2h\nXeK4VMxiKbHne4KdoPbQgk8m83BniAOWOKOTab
9wFjwi+1wd875jV3r1SS64uz+9\ncJ6Pp+4Kv+cvjZyMWPTnSKI4QoXlbQd+92lXZwqtGCTvzAPdLm5E/B8ber2/lb2s\nwJ4f8A7jeQs5VtDZmVoeXLd8Sdm/ocji4+pGqQIDAQ
ABozIwMDAPBgNVHRMBAf8E\nBTADAQH/MB0GA1UdDgQWBBQyiqUipUYOXyyDknCF/imtYlYyLzANBgkqhkiG9w0B\nAQsFAAOCAQEAYfoiiBX5fNJRpm/MW1VVdE62f7K+3OatTP
oroZa6CTgULZhdTnWG\nWP5MIZt9iKq3AQzgma7aQEt/dKJJL3nmDqsY/fHRjjjcF6SSu1W4Fp32BFp3mi57\naeICvFqMgHztyBB5WJVsU7EktWSn0PjWSgcBWticXFw8aOG55H
S5rrUqi9mBYWSm\njo/7rVsX/UOkUHP19sBu6JrJarvibkUdyxY66mOEpLMmf9My/11TIL5s2rhpqRSX\nR9DsiyGAq7KXaJzMsxWn1DxvlyRfjNo3R3W0ICWq1F6y6RtwU+ofAo
TxMbC25XQ0\ncYI/qnKkj0/K4ycY0QPZN7qr76I5INP5iA==\n-----END CERTIFICATE-----\n'])])
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/alert: {"message": "Production Anomaly Detected. Check the Production Plant!", "sequence": 0}

Publish received on topic clients/CMPE_Test_Client1/alert
b'{"message": "Production Anomaly Detected. Check the Production Plant!", "sequence": 0}'
```

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

Verify Process on Core Device’s Terminal

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

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

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

Publish Topic to Trigger Subscribers (AWS IoT Core)

- Publish the topic with basic_discovery.py script.

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

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

This message will be relayed to AWS IoT Core

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

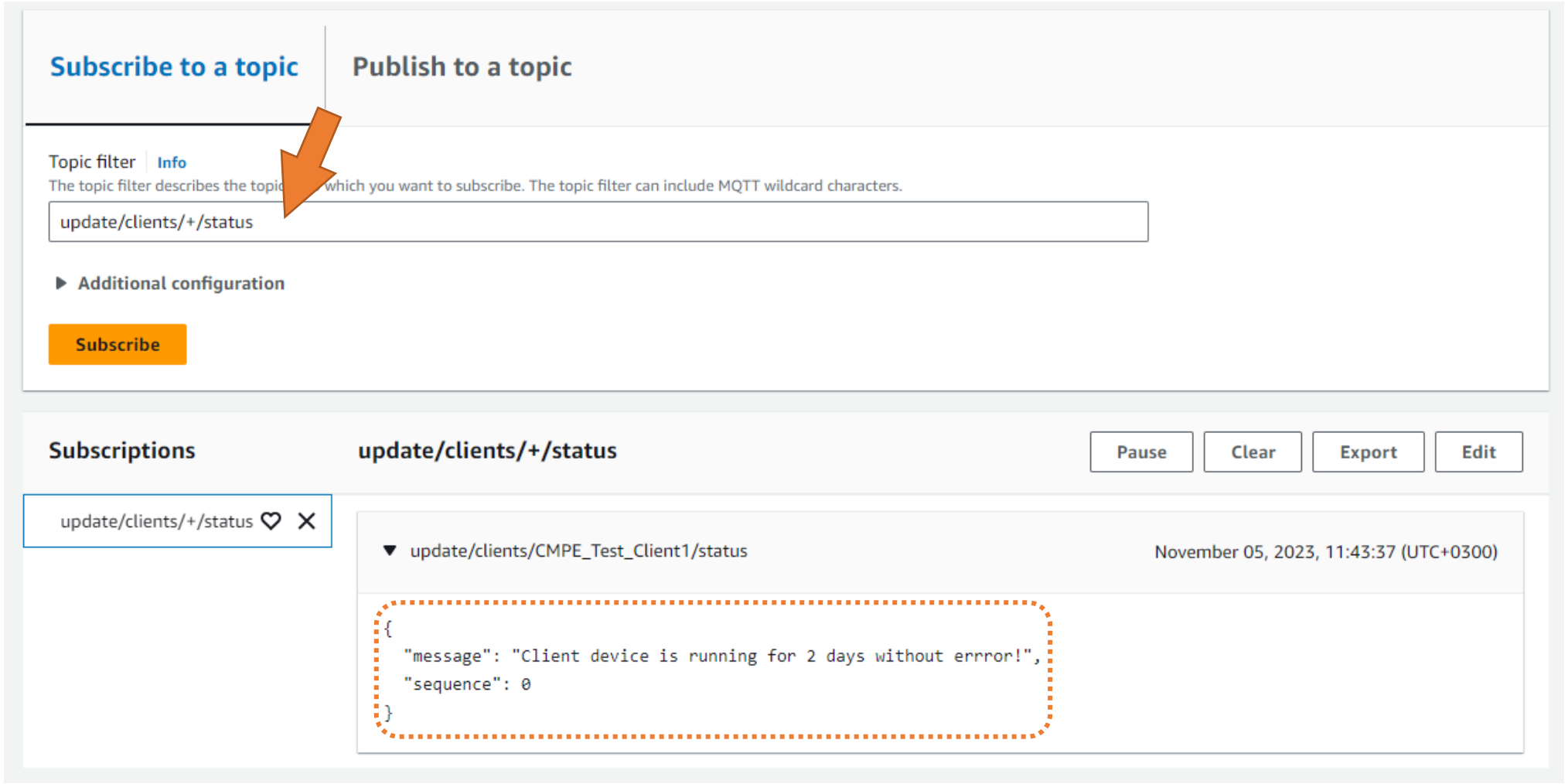
Step 5: Publish 'clients/+/status' Topic From IoT Client

Verify Process on Terminal

```
cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
    --thing_name CMPE_Test_Client1 \
    --topic 'clients/CMPE_Test_Client1/status' \
    --message 'Client device is running for 2 days without errror!' \
    --ca_file /home/cagatay/shared_folder/AmazonRootCA1.pem \
    --cert /home/cagatay/shared_folder/certificate.pem.crt \
    --key /home/cagatay/shared_folder/private.pem.key \
    --region eu-central-1 \
    --verbosity Warn \
   --max_pub_ops 1
Performing greengrass discovery...
[WARN] [2023-11-05T08:41:03Z] [00007f61829fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevice-CMPE583
-GreengrassCore', cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCor
e', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address='10.0.2.15', metadata='', port=8883)])], cer
tificate_authorities=['-----BEGIN CERTIFICATE-----\nMIID1TCCAr2gAwIBAgIVAMDxNyNBeW2Jja8e093NSaPTIV8WMA0GCSqGSIb3DQEB\nCwUAMIGJMQswCQYDVQ
QGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBJbmMuMRww\nGgYDVQQLDBNBbWF6b24gV2ViIFNlcnZpY2VzMRMwEQYDVQQIDApXYXNoaW5ndG9u\nMRAwDgYDVQQHDAdTZWF0dG
xlMRswGQYDVQQDDBJHcmVlbmdyYXNzIENvcmUgQ0Ew\nHhcNMjMxMTA0MTMwMTM5WhcNMjgxMTAyMTMwMTM5WjCBiTELMAkGA1UEBhMCVVMx\nGDAWBgNVBAoMD0FtYXpvbi5jb2
0gSW5jLjEcMBoGA1UECwwTQW1hem9uIFdlYiBT\nZXJ2aWNlczETMBEGA1UECAwKV2FzaGluZ3RvbjEQMA4GA1UEBwwHU2VhdHRsZTEb\nMBkGA1UEAwwSR3JlZW5ncmFzcyBDb3
JlIENBMIIBIjANBgkqhkiG9w0BAQEFAAOC\nAQ8AMIIBCgKCAQEAmZ2lfIUSehjcUmSgkm1uCt8VoKcQtWIoy0pzVmyNaR2RnDFr\n48huzRAT+Ee6dVxyLMRcsw1ncJM62cPv8w
ATNDgC/IMdtotLLqwAGKdcFfFiWvtx\nEZ1zfN8JsQJ8Mutr4X+PfFmFSqxSTdfTXB2l7Z0SX0SE7auaLzHcH8HTDYAklp2h\nXeK4VMxiKbHne4KdoPbQgk8m83BniAOWOKOTab
9wFjwi+1wd875jV3r1SS64uz+9\ncJ6Pp+4Kv+cvjZyMWPTnSKI4QoXlbQd+92lXZwqtGCTvzAPdLm5E/B8ber2/lb2s\nwJ4f8A7jeQs5VtDZmVoeXLd8Sdm/ocji4+pGqQIDAQ
ABozIwMDAPBgNVHRMBAf8E\nBTADAQH/MB0GA1UdDgQWBBQyiqUipUYOXyyDknCF/imtYlYyLzANBgkqhkiG9w0B\nAQsFAAOCAQEAYfoiiBX5fNJRpm/MW1VVdE62f7K+3OatTP
oroZa6CTgULZhdTnWG\nWP5MIZt9iKq3AQzgma7aQEt/dKJJL3nmDqsY/fHRjjjcF6SSu1W4Fp32BFp3mi57\naeICvFqMgHztyBB5WJVsU7EktWSn0PjWSgcBWticXFw8aOG55H
S5rrUqi9mBYWSm\njo/7rVsX/UOkUHP19sBu6JrJarvibkUdyxY66mOEpLMmf9My/11TIL5s2rhpqRSX\nR9DsiyGAq7KXaJzMsxWn1DxvlyRfjNo3R3W0ICWq1F6y6RtwU+ofAo
TxMbC25XQ0\ncYI/qnKkj0/K4ycY0QPZN7qr76I5INP5iA==\n-----END CERTIFICATE-----\n'])])
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/status: {"message": "Client device is running for 2 days without errror!", "sequence": 0}

Publish received on topic clients/CMPE_Test_Client1/status
b'{"message": "Client device is running for 2 days without errror!", "sequence": 0}'
```

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

Verify Process on MQTT Test Client



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

Subscribe Topic From Test Client 2

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

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

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

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

Publish Topic to Trigger Subscribers (Test Client 2)

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

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

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

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

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

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

Verify Topic Published

```
(venvclient1) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
    --thing_name 'CMPE_Test_Client1' \
    --topic 'clients/CMPE_Test_Client1/sync' \
    --message 'Hello from IoT Client 1!' \
    --ca_file ~/shared_folder/AmazonRootCA1.pem \
    --cert ~/shared_folder/certificate.pem.crt \
    --key ~/shared_folder/private.pem.key \
    --region eu-central-1 \
    --verbosity Warn \
   --max_pub_ops 1 \
   --mode publish
Performing greengrass discovery...
[WARN] [2023-11-06T13:52:20Z] [00007fd3ac8fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDevic
22364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_address
AwIBAgIVAJS5wdMBGrA7mIHUCs183HgzuqAnMA0GCSqGSIb3DQEB\nCwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBJbmMuMRww\nGgYDVQQ
NzIENvcmUgQ0Ew\nHhcNMjMxMTA2MTE1ODE0WhcNMjgxMTA0MTE1ODE0WjCBiTELMAkGA1UEBhMCVVMx\nGDAWBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMBoGA1UEC
FzcyBDb3JlIENBMIIBIjANBgkqhkiG9w0BAQEFAAOC\nAQ8AMIIBCgKCAQEAsfbYoDMM56kG2MLGqo3/6RUFDquIN97/4qQ/CD727F45WHPd\nuwbBtACsSXXSvc7F3
k1Z3\nwhEobHrDR8hFmjBQ5y1uOWj3mIpScCELAtudjScHHQVOdbcium+fOvn+uOhIVOxO\nYydkscLyVyPYcL0h7D7B0n1hsUv3WTOgYcmh1zSN49/29D9UdG07i0f
rWQrz9sINS7QZLNS1zANBgkqhkiG9w0B\nAQsFAAOCAQEAMd5Pmw42KOZWvV9ZAM1W3g2D4egNa98jBogP8otc4dOg3vhtFtVF\n+jWWF5kinHsMVvOuQTjxV6Rf17F
iYCRts3xobU47Xo5w2nl4Z+IuCJGeT3LKm1arGR4is4c2SpImAdUkjEU8SGh\nHEkifLNFejNBx/wurqPyPRk8yw6R9c1j7ExHFHlYRLpzBgUfcy+pNDUipE6A4jHpV
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/sync: {"message": "Hello from IoT Client 1!", "sequence": 0}
```

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

Verify Topic Received

```
(venvclient2) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
    --thing_name 'CMPE_Test_Client2' \
    --topic 'clients/+/sync' \
    --ca_file ~/shared_folder/AmazonRootCA1.pem \
    --cert ~/shared_folder/client2-certificate.pem.crt \
    --key ~/shared_folder/client2-private.pem.key \
    --region eu-central-1 \
    --verbosity Warn \
   --max_pub_ops 1 \
   --mode subscribe
Performing greengrass discovery...
[WARN] [2023-11-06T13:51:22Z] [00007fd0277fe640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDe
22364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_addr
AwIBAgIVAJS5wdMBGrA7mIHUCs183HgzuqAnMA0GCSqGSIb3DQEB\nCwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBJbmMuMRww\nGgYD
NzIENvcmUgQ0Ew\nHhcNMjMxMTA2MTE1ODE0WhcNMjgxMTA0MTE1ODE0WjCBiTELMAkGA1UEBhMCVVMx\nGDAWBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMBoGA1U
FzcyBDb3JlIENBMIIBIjANBgkqhkiG9w0BAQEFAAOC\nAQ8AMIIBCgKCAQEAsfbYoDMM56kG2MLGqo3/6RUFDquIN97/4qQ/CD727F45WHPd\nuwbBtACsSXXSvc
k1Z3\nwhEobHrDR8hFmjBQ5y1uOWj3mIpScCELAtudjScHHQVOdbcium+fOvn+uOhIVOxO\nYydkscLyVyPYcL0h7D7B0n1hsUv3WTOgYcmh1zSN49/29D9UdG07
rWQrz9sINS7QZLNS1zANBgkqhkiG9w0B\nAQsFAAOCAQEAMd5Pmw42KOZWvV9ZAM1W3g2D4egNa98jBogP8otc4dOg3vhtFtVF\n+jWWF5kinHsMVvOuQTjxV6Rf
iYCRts3xobU47Xo5w2nl4Z+IuCJGeT3LKm1arGR4is4c2SpImAdUkjEU8SGh\nHEkifLNFejNBx/wurqPyPRk8yw6R9c1j7ExHFHlYRLpzBgUfcy+pNDUipE6A4j
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Publish received on topic clients/CMPE_Test_Client1/sync
b'{"message": "Hello from IoT Client 1!", "sequence": 0}'
```

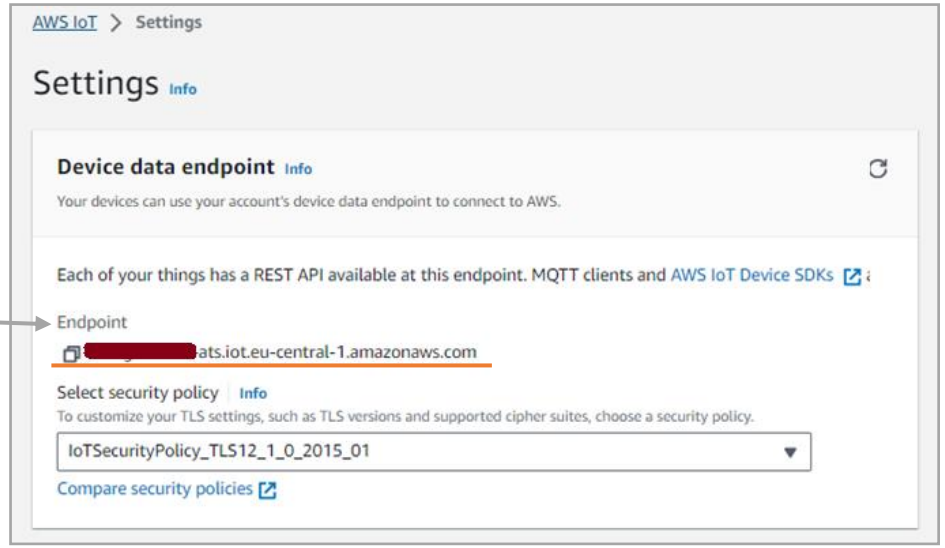
Receiving Events From AWS IoT Core

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

✓ Use an empty--message with pubsub.py script to 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 settings page of AWS IoT Console



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

Subscribe Topic From Test Client 2

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

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

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

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

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

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

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

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

This message will be relayed to AWS IoT Core

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

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

Verify Topic Published

```
(venvclient1) cagatay@kubuntu:~/Projects/greengrass-demo/aws-iot-device-sdk-python-v2/samples$ python3 basic_discovery.py \
    --thing_name 'CMPE_Test_Client1' \
    --topic 'clients/CMPE_Test_Client1/status' \
    --message 'Status update from Client 1!' \
    --ca_file ~/shared_folder/AmazonRootCA1.pem \
    --cert ~/shared_folder/certificate.pem.crt \
    --key ~/shared_folder/private.pem.key \
    --region eu-central-1 \
    --verbosity Warn \
   --max_pub_ops 1 \
   --mode publish
Performing greengrass discovery...
[WARN] [2023-11-06T15:00:26Z] [00007f511e1fd640] [http-connection] - static: Unrecognized ALPN protocol. Assuming HTTP/1.1
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_id='greengrassV2-coreDev
22364162:thing/CMPE583-GreengrassCore', connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='10.0.2.15', host_addre
AwIBAgIVAJS5wdMBGrA7mIHUCs183HgzuqAnMA0GCSqGSIb3DQEB\nCwUAMIGJMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBJbmMuMRww\nGgYDV
NzIENvcmUgQ0Ew\nHhcNMjMxMTA2MTE1ODE0WhcNMjgxMTA0MTE1ODE0WjCBiTELMAkGA1UEBhMCVVMx\nGDAWBgNVBAoMD0FtYXpvbi5jb20gSW5jLjEcMBoGA1U
FzcyBDb3JlIENBMIIBIjANBgkqhkiG9w0BAQEFAAOC\nAQ8AMIIBCgKCAQEAsfbYoDMM56kG2MLGqo3/6RUFDquIN97/4qQ/CD727F45WHPd\nuwbBtACsSXXSvc7
k1Z3\nwhEobHrDR8hFmjBQ5y1uOWj3mIpScCELAtudjScHHQVOdbcium+fOvn+uOhIVOxO\nYydkscLyVyPYcL0h7D7B0n1hsUv3WTOgYcmh1zSN49/29D9UdG07i
rWQrz9sINS7QZLNS1zANBgkqhkiG9w0B\nAQsFAAOCAQEAMd5Pmw42KOZWvV9ZAM1W3g2D4egNa98jBogP8otc4dOg3vhtFtVF\n+jWWF5kinHsMVvOuQTjxV6Rf1
iYCRts3xobU47Xo5w2nl4Z+IuCJGeT3LKm1arGR4is4c2SpImAdUkjEU8SGh\nHEkifLNFejNBx/wurqPyPRk8yw6R9c1j7ExHFHlYRLpzBgUfcy+pNDUipE6A4jH
Trying core arn:aws:iot:eu-central-1:465822364162:thing/CMPE583-GreengrassCore at host 10.0.2.15 port 8883
Connected!
Published topic clients/CMPE_Test_Client1/status: {"message": "Status update from Client 1!", "sequence": 0}
```

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

Verify Topic Received

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

Thank You