Edge Computing with AWS IoT Greengrass

Çağatay Sönmez

03.11.2023

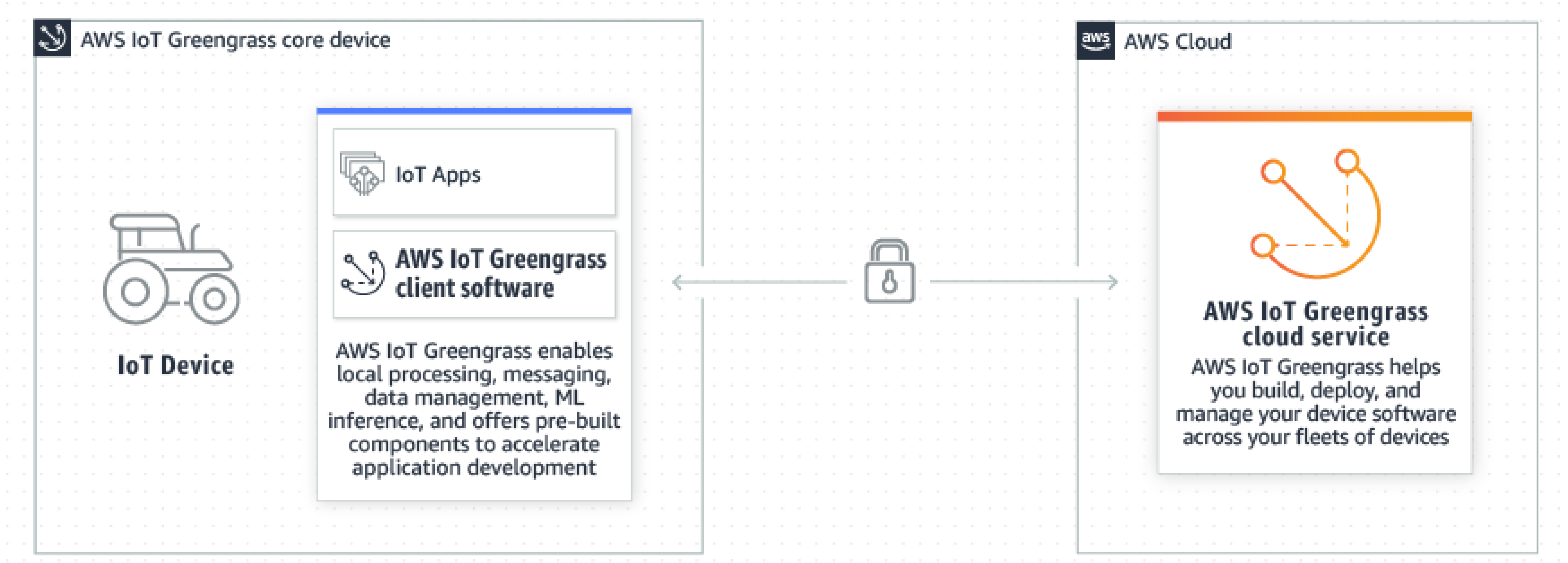
The Primary Sources Used in This Lecture

- 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

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

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].


AWS IoT Greengrass core device
IoT Device
IoT Apps
AWS IoT Greengrass client software
AWS IoT Greengrass enables local processing, messaging, data management, ML inference, and offers pre-built components to accelerate application development
AWS Cloud
AWS IoT Greengrass cloud service
AWS IoT Greengrass helps you build, deploy, and manage your device software across your fleets of devices

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

AWS IoT Greengrass Key Concepts

Greengrass Core Device

Greengrass Client Device

Greengrass Component

Deployment

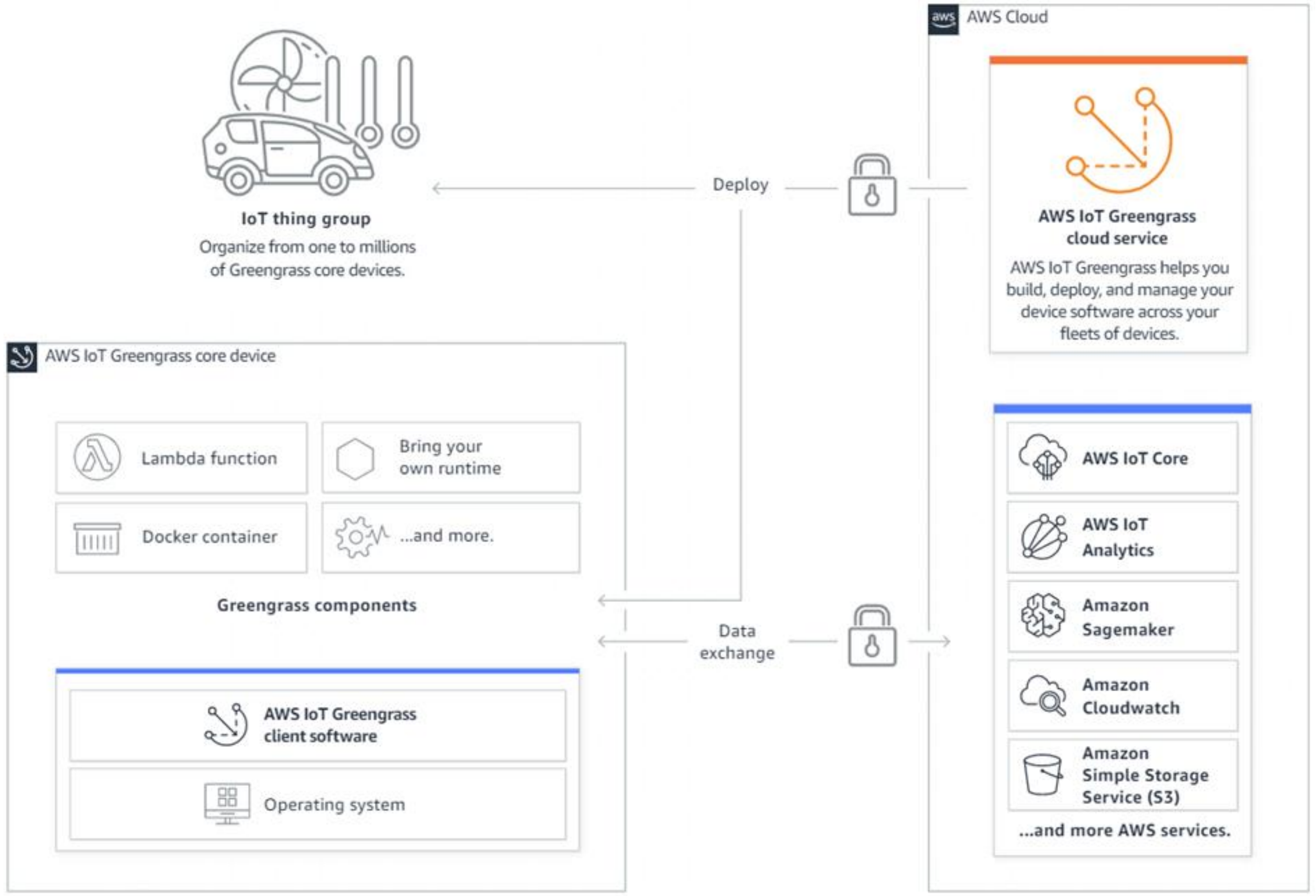
Greengrass Core Software

How AWS IoT Greengrass works

Greengrass core device is an AWS IoT thing (device) that acts as a hub or gateway in edge environments.

Greengrass component is software module that is deployed to and runs on the core device.

Greengrass core (client) software is a set of all Greengrass components that you install on a core device.



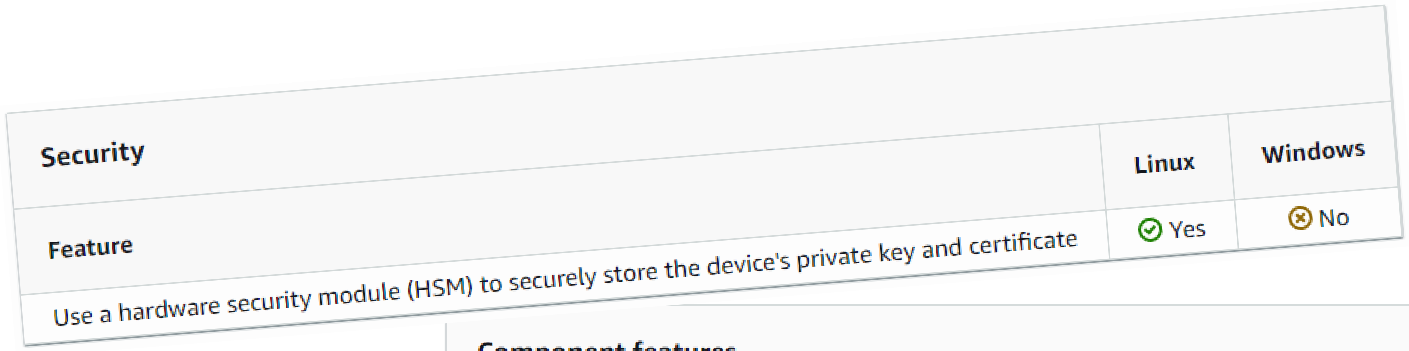
AWS Cloud
IoT thing group
Organize from one to millions of Greengrass core devices.
Deploy
AWS IoT Greengrass cloud service
AWS IoT Greengrass helps you build, deploy, and manage your device software across your fleets of devices.
AWS IoT Greengrass core device
Lambda function
Bring your own runtime
Docker container
...and more.
Greengrass components
Data exchange
AWS IoT Greengrass client software
Operating system
AWS IoT Core
AWS IoT Analytics
Amazon Sagemaker
Amazon Cloudwatch
Amazon Simple Storage Service (S3)
...and more AWS services.

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…

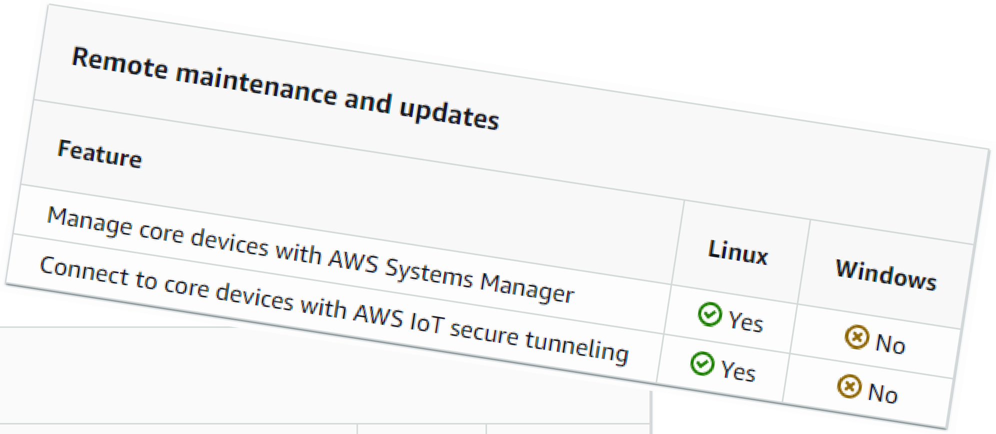
Selecting Client’s Operating System

- Some features are supported on only certain operating systems!



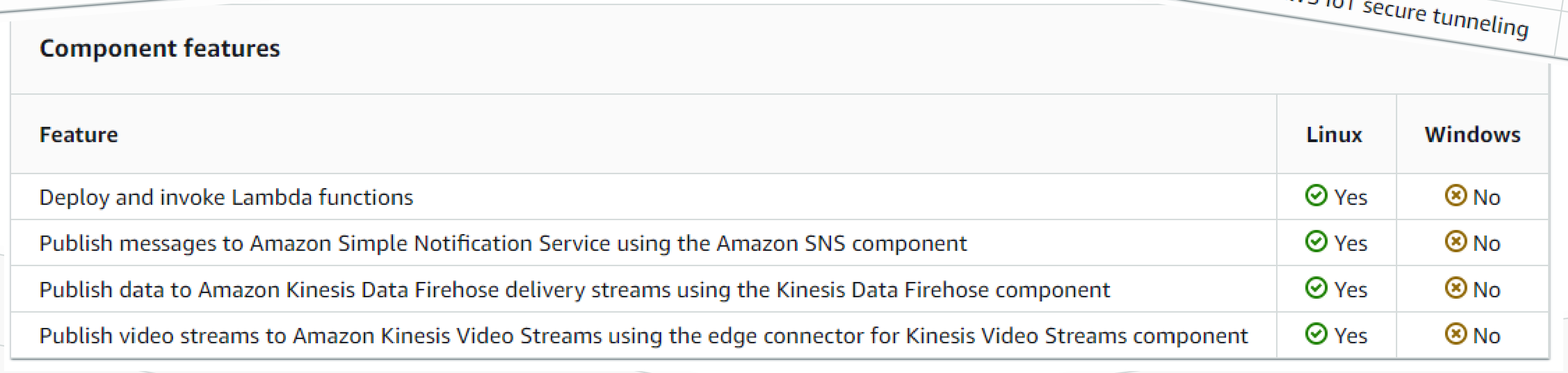
Security

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



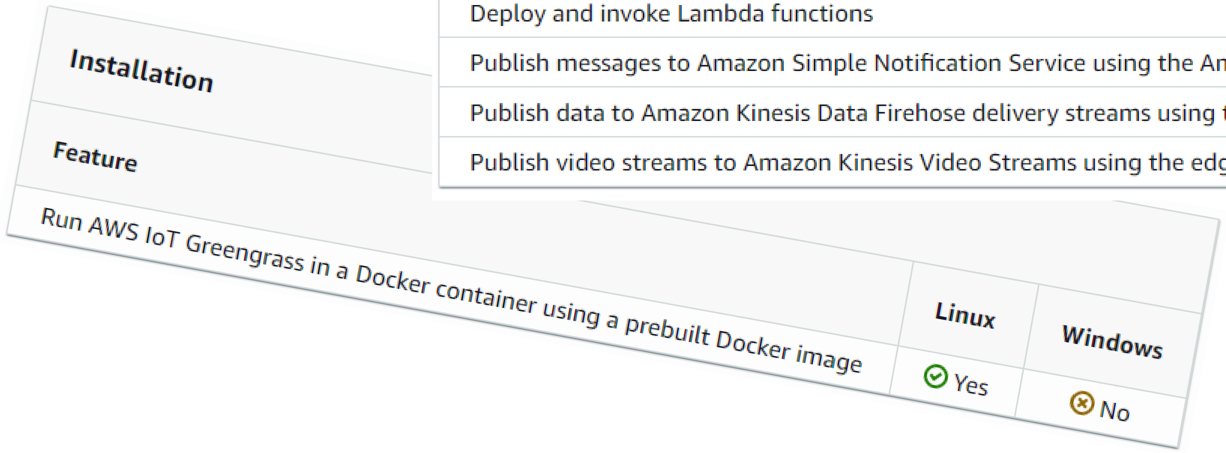
Remote maintenance and updates

Feature	Linux	Windows
Manage core devices with AWS Systems Manager	✓ Yes	⊗ No
Connect to core devices with AWS IoT secure tunneling	✓ Yes	⊗ No



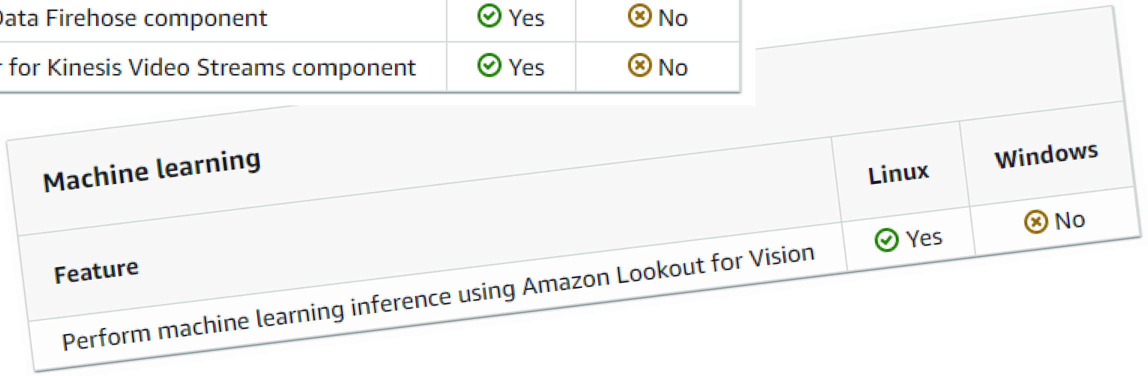
Component features

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

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
```

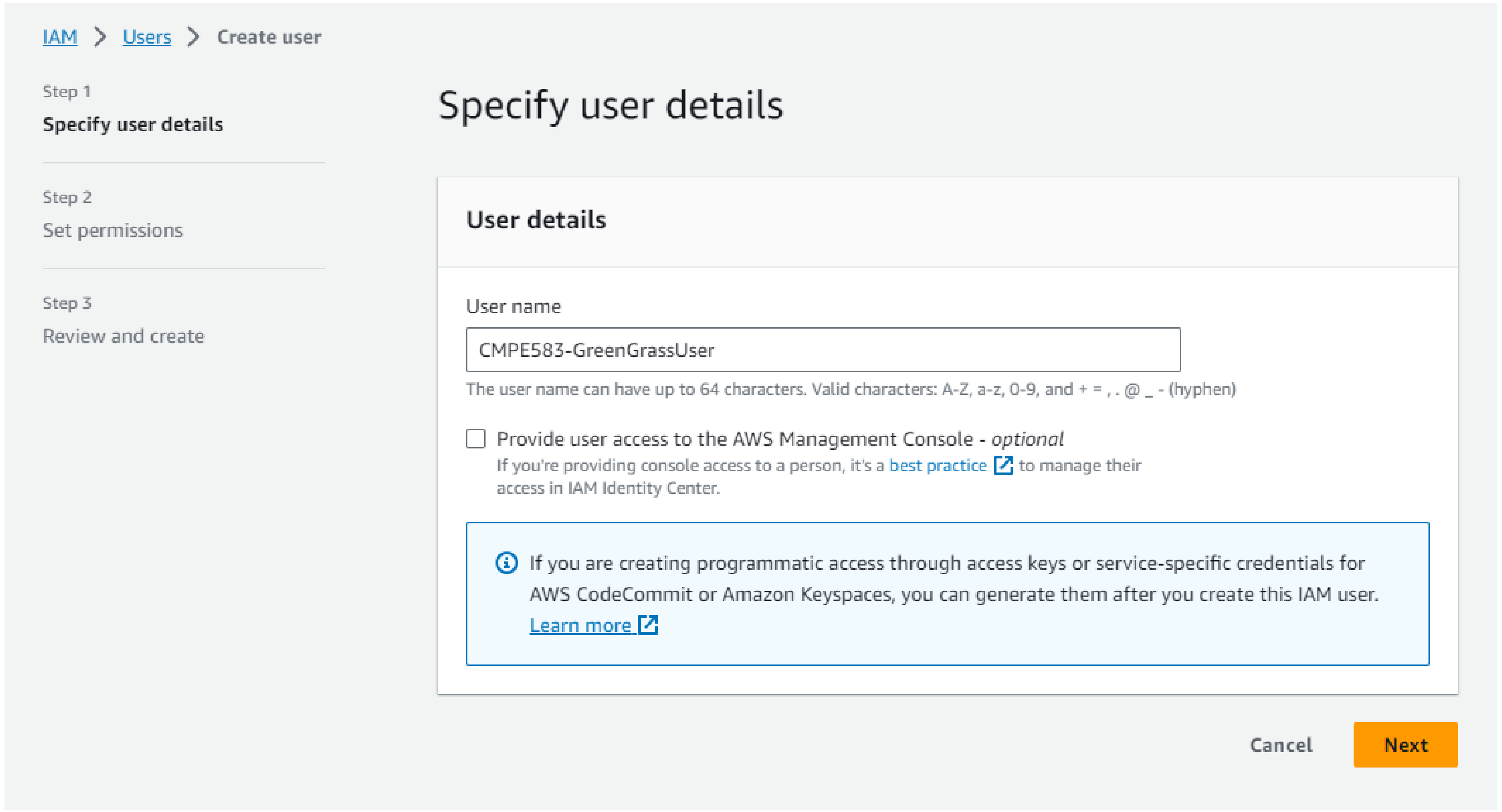
For Red Hat-based distributions:

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

Step 2: Create AWS Credentials

Create a Temporary User

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



IAM > Users > Create user

Step 1
Specify user details

Step 2
Set permissions

Step 3
Review and create

Specify user details

User details

User name
CMPE583-GreenGrassUser

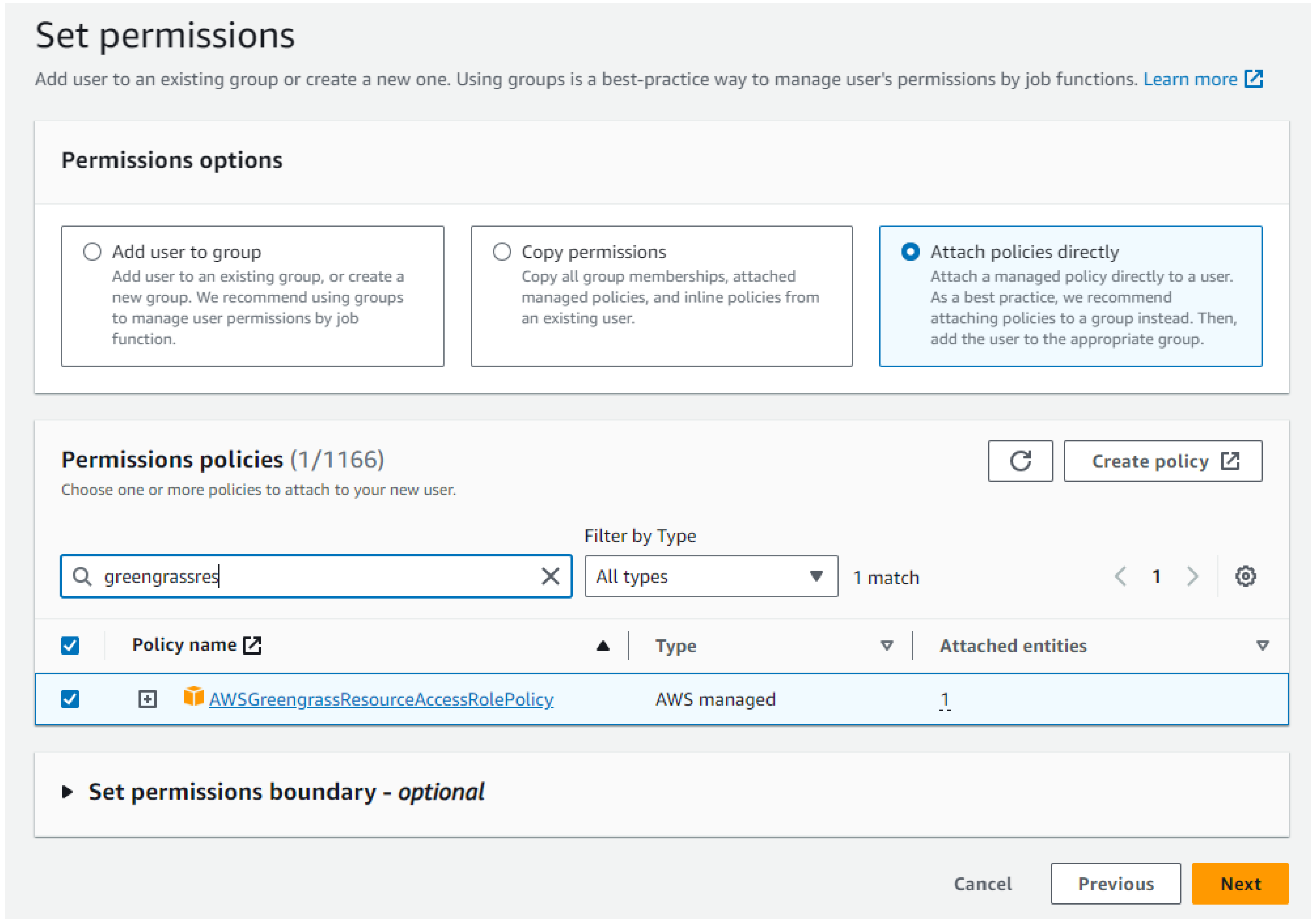
The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

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

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

Step 2: Create AWS Credentials

Attach Required Policies to the User



Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

- ○ Add user to group
  Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- ○ Copy permissions
  Copy all group memberships, attached managed policies, and inline policies from an existing user.
- ● Attach policies directly
  Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1166)

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

Create policy

Filter by Type

greengrassres | All types | 1 match

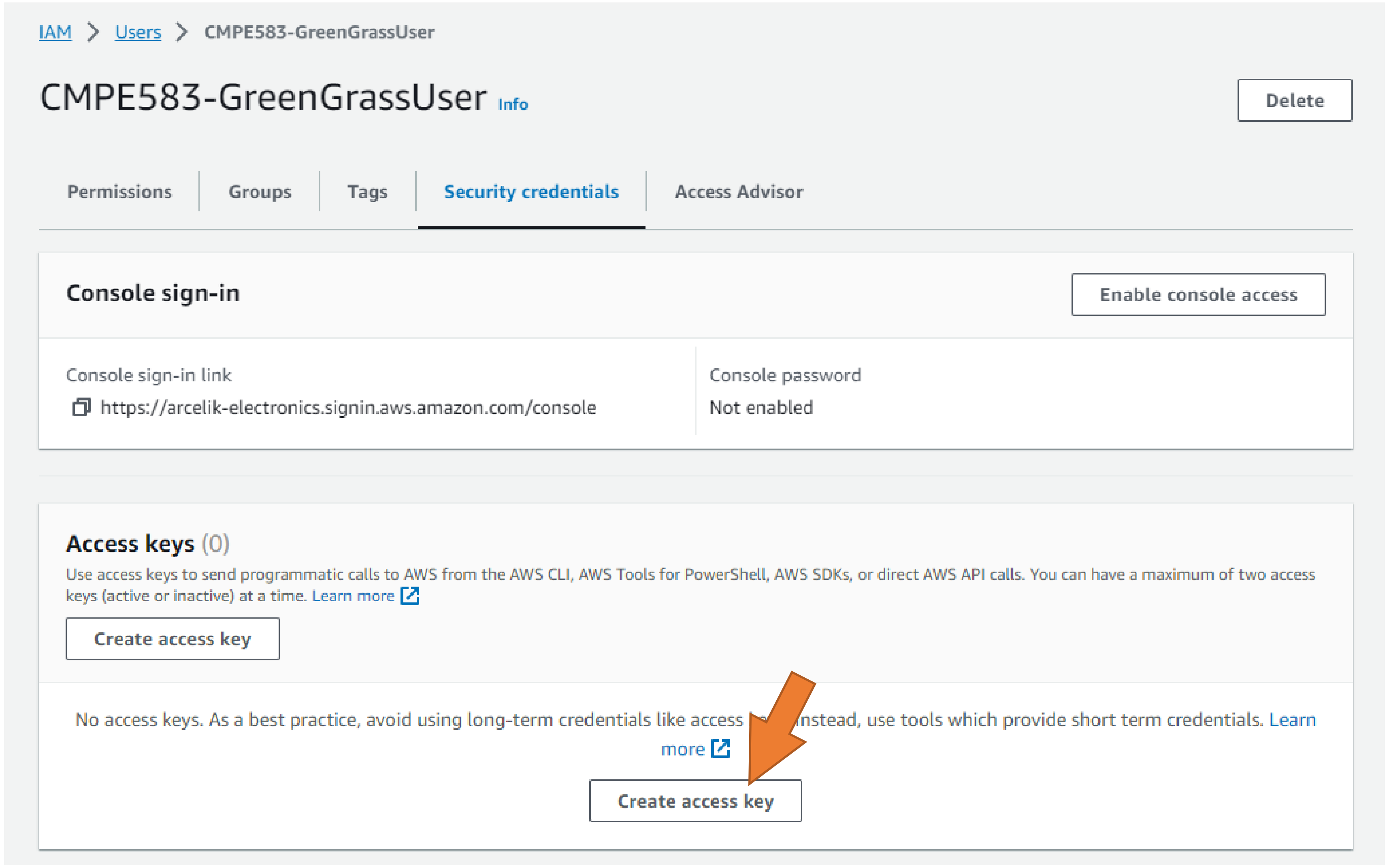
✓	Policy name	Type	Attached entities
✓	AWSGreengrassResourceAccessRolePolicy	AWS managed	1

▸ **Set permissions boundary - *optional***

Cancel | Previous | Next

Step 2: Create AWS Credentials

Create Access Keys for the User



IAM > Users > CMPE583-GreenGrassUser

CMPE583-GreenGrassUser Info

Console sign-in

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

Console password
Not enabled

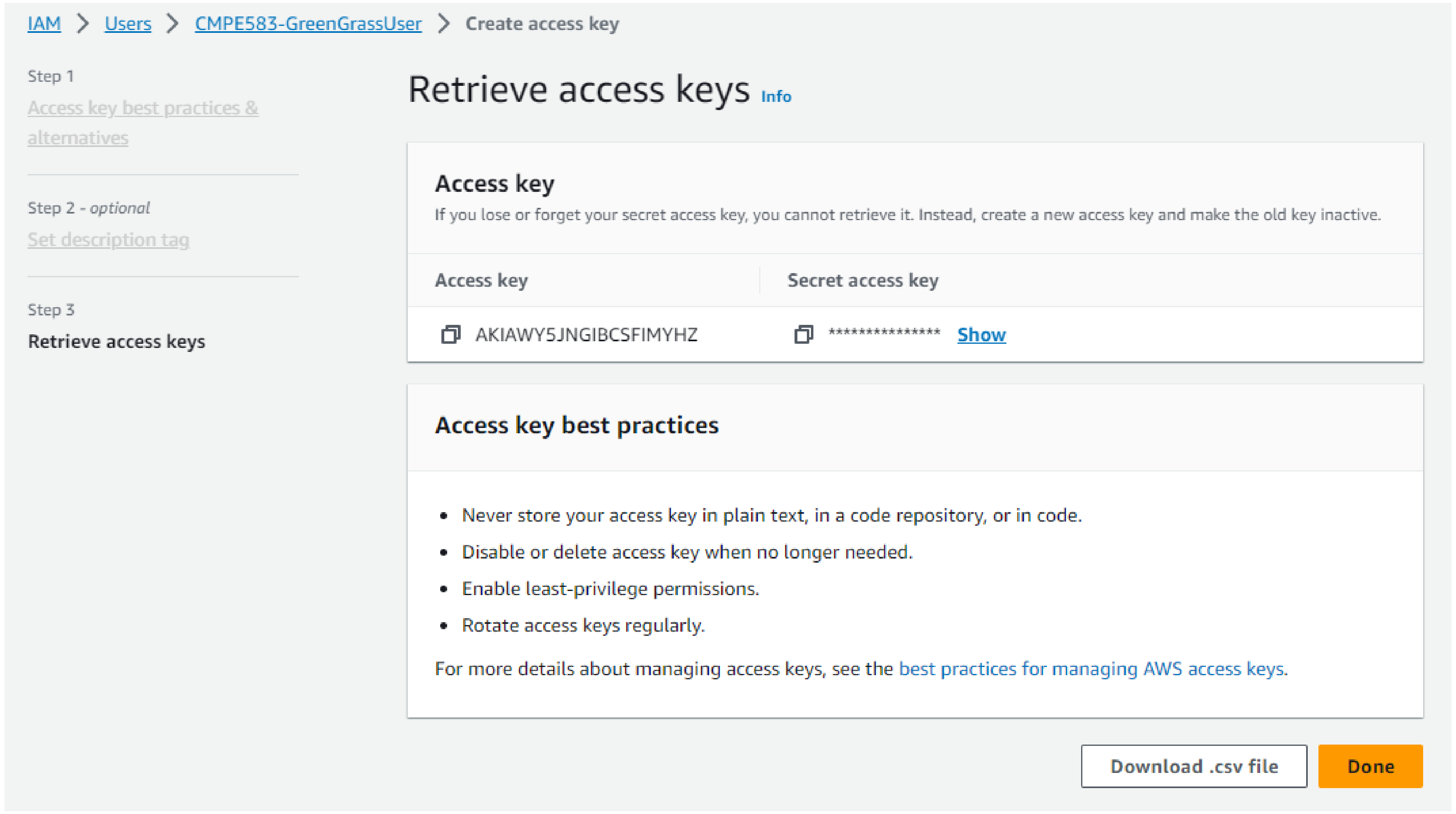
Access keys (0)

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

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

Step 2: Create AWS Credentials

Copy the Access Keys



IAM > Users > CMPE583-GreenGrassUser > Create access key

Step 1
Access key best practices & alternatives

Step 2 - *optional*
Set description tag

Step 3
Retrieve access keys

Retrieve access keys Info

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

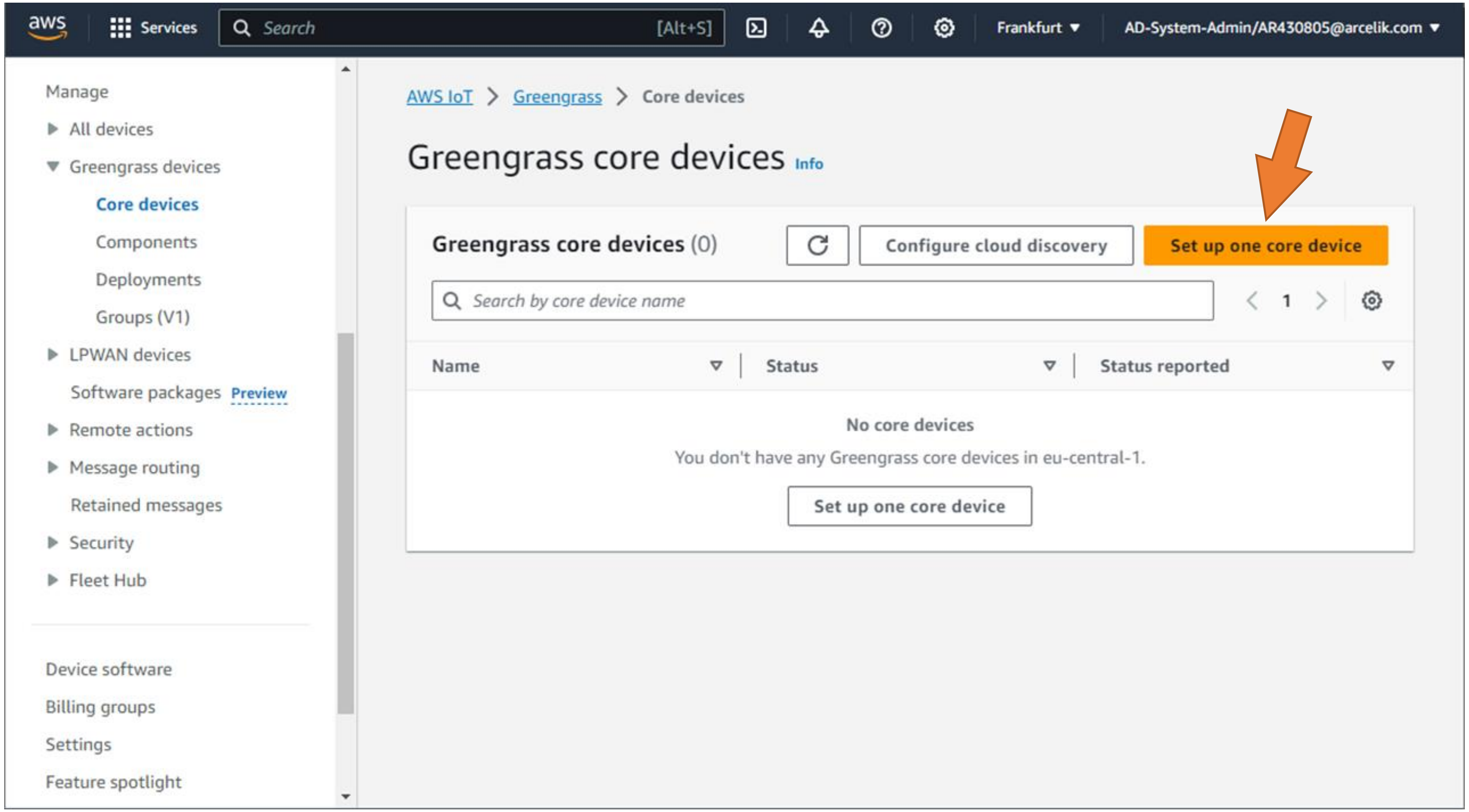
Access key	Secret access key
AKIAWY5JNGIBCSFIMYHZ	*************** Show

Access key best practices

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

For more details about managing access keys, see the best practices for managing AWS access keys.

Step 3: Set Up a Greengrass Core Device in AWS



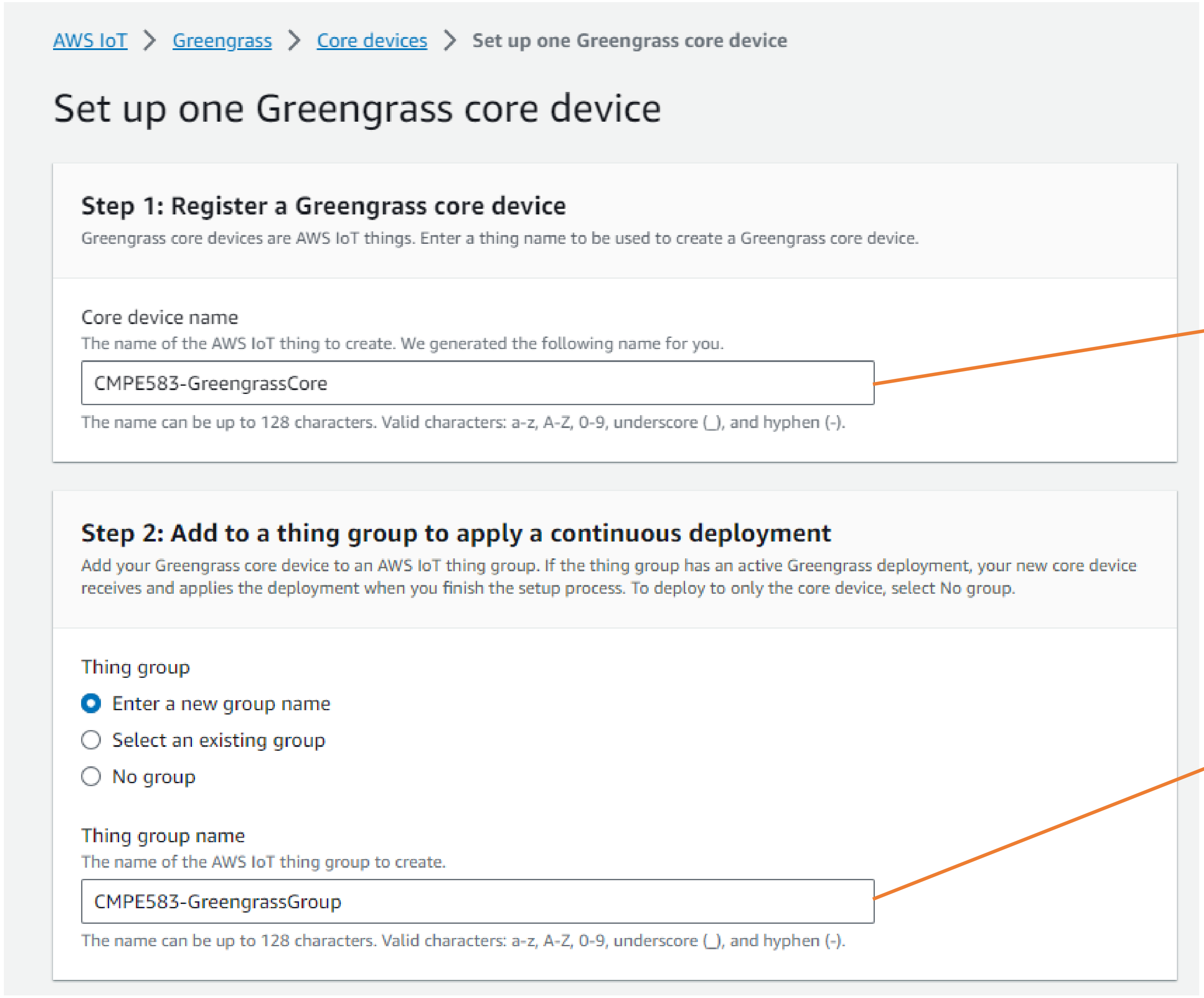
Greengrass core devices

Greengrass core devices (0)

No core devices

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

Step 3: Set Up a Greengrass Core Device in AWS



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

Set up one Greengrass core device

Step 1: Register a Greengrass core device

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

Core device name

The name of the AWS IoT thing to create. We generated the following name for you.

CMPE583-GreengrassCore

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

Step 2: Add to a thing group to apply a continuous deployment

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

Thing group

- [✓ Enter a new group name]
- Select an existing group
- No group

Thing group name

The name of the AWS IoT thing group to create.

CMPE583-GreengrassGroup

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

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

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

Step 4: Install the Greengrass Core Software

Configure AWS Credentials on the Client

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

  Provide credentials as environment variables before running the installer:

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

- Credentials are not saved by the Greengrass installer!

Step 4: Install the Greengrass Core Software

For Linux Based Devices

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

Download the installer:

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

Run the installer:

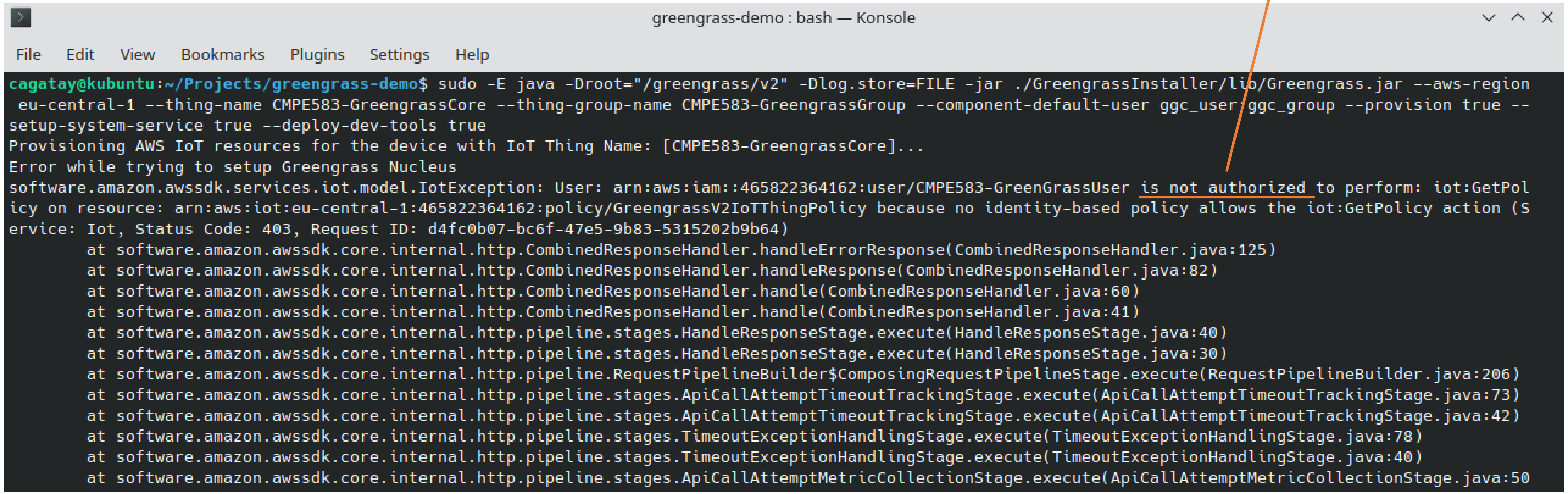
```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE -jar ./GreengrassInstaller/lib/Greengrass.jar --aws-
        region eu-central-1 --thing-name CMPE583-GreengrassCore --thing-group-name CMPE583-
        GreengrassGroup --component-default-user ggc_user:ggc_group --provision true --setup-system-service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

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

Step 4: Install the Greengrass Core Software

Run the Installer Command

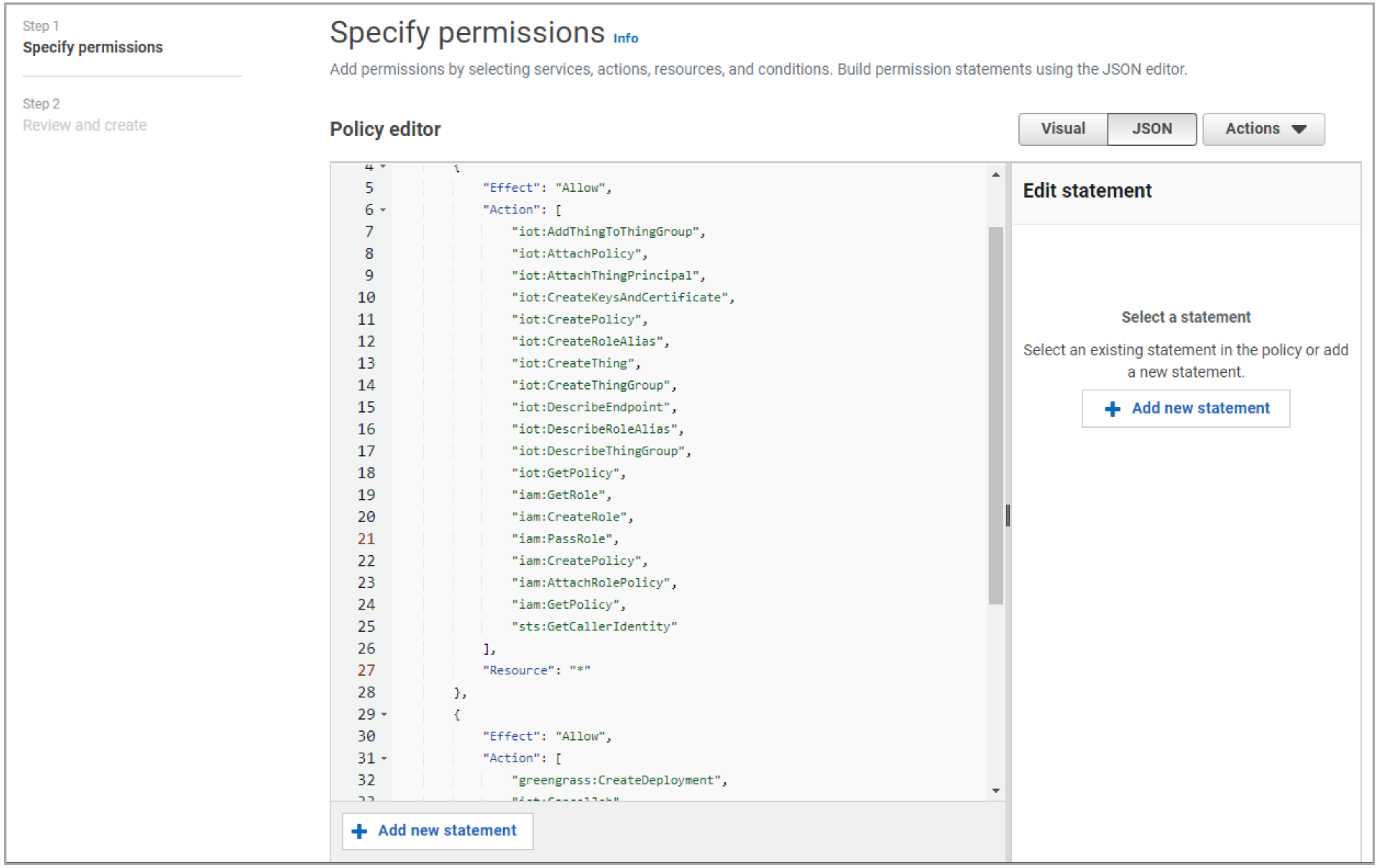
The user is not authorized to perform installer jobs! You should attach required permissions!


greengrass-demo : bash — Konsole
File Edit View Bookmarks Plugins Settings Help

```
cagatay@kubuntu:~/Projects/greengrass-demo$ sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE -jar ./GreengrassInstaller/lib/Greengrass.jar --aws-region
 eu-central-1 --thing-name CMPE583-GreengrassCore --thing-group-name CMPE583-GreengrassGroup --component-default-user ggc_user:ggc_group --provision true --
setup-system-service true --deploy-dev-tools true
Provisioning AWS IoT resources for the device with IoT Thing Name: [CMPE583-GreengrassCore]...
Error while trying to setup Greengrass Nucleus
software.amazon.awssdk.services.iot.model.IotException: User: arn:aws:iam::465822364162:user/CMPE583-GreenGrassUser is not authorized to perform: iot:GetPol
icy on resource: arn:aws:iot:eu-central-1:465822364162:policy/GreengrassV2IoTThingPolicy because no identity-based policy allows the iot:GetPolicy action (S
ervice: Iot, Status Code: 403, Request ID: d4fc0b07-bc6f-47e5-9b83-5315202b9b64)
        at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handleErrorResponse(CombinedResponseHandler.java:125)
        at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handleResponse(CombinedResponseHandler.java:82)
        at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handle(CombinedResponseHandler.java:60)
        at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handle(CombinedResponseHandler.java:41)
        at software.amazon.awssdk.core.internal.http.pipeline.stages.HandleResponseStage.execute(HandleResponseStage.java:40)
        at software.amazon.awssdk.core.internal.http.pipeline.stages.HandleResponseStage.execute(HandleResponseStage.java:30)
        at software.amazon.awssdk.core.internal.http.pipeline.RequestPipelineBuilder$ComposingRequestPipelineStage.execute(RequestPipelineBuilder.java:206)
        at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptTimeoutTrackingStage.execute(ApiCallAttemptTimeoutTrackingStage.java:73)
        at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptTimeoutTrackingStage.execute(ApiCallAttemptTimeoutTrackingStage.java:42)
        at software.amazon.awssdk.core.internal.http.pipeline.stages.TimeoutExceptionHandlingStage.execute(TimeoutExceptionHandlingStage.java:78)
        at software.amazon.awssdk.core.internal.http.pipeline.stages.TimeoutExceptionHandlingStage.execute(TimeoutExceptionHandlingStage.java:40)
        at software.amazon.awssdk.core.internal.http.pipeline.stages.ApiCallAttemptMetricCollectionStage.execute(ApiCallAttemptMetricCollectionStage.java:50
```

Step 4: Install the Greengrass Core Software

Set Required Permissions to Greengrass User



Step 1
Specify permissions

Step 2
Review and create

Specify permissions Info

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

Policy editor

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy",
                "iam:GetRole",
                "iam:CreateRole",
                "iam:PassRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "iam:GetPolicy",
                "sts:GetCallerIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
```

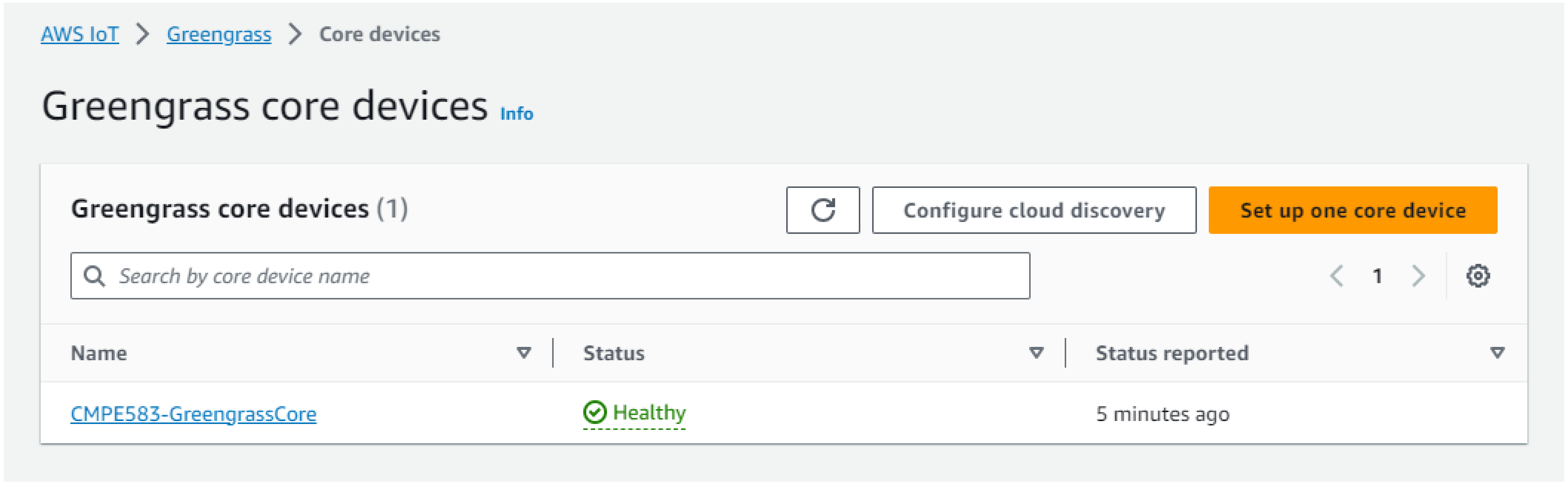
Edit statement

Select a statement

Select an existing statement in the policy or add a new statement.

Step 4: Install the Greengrass Core Software

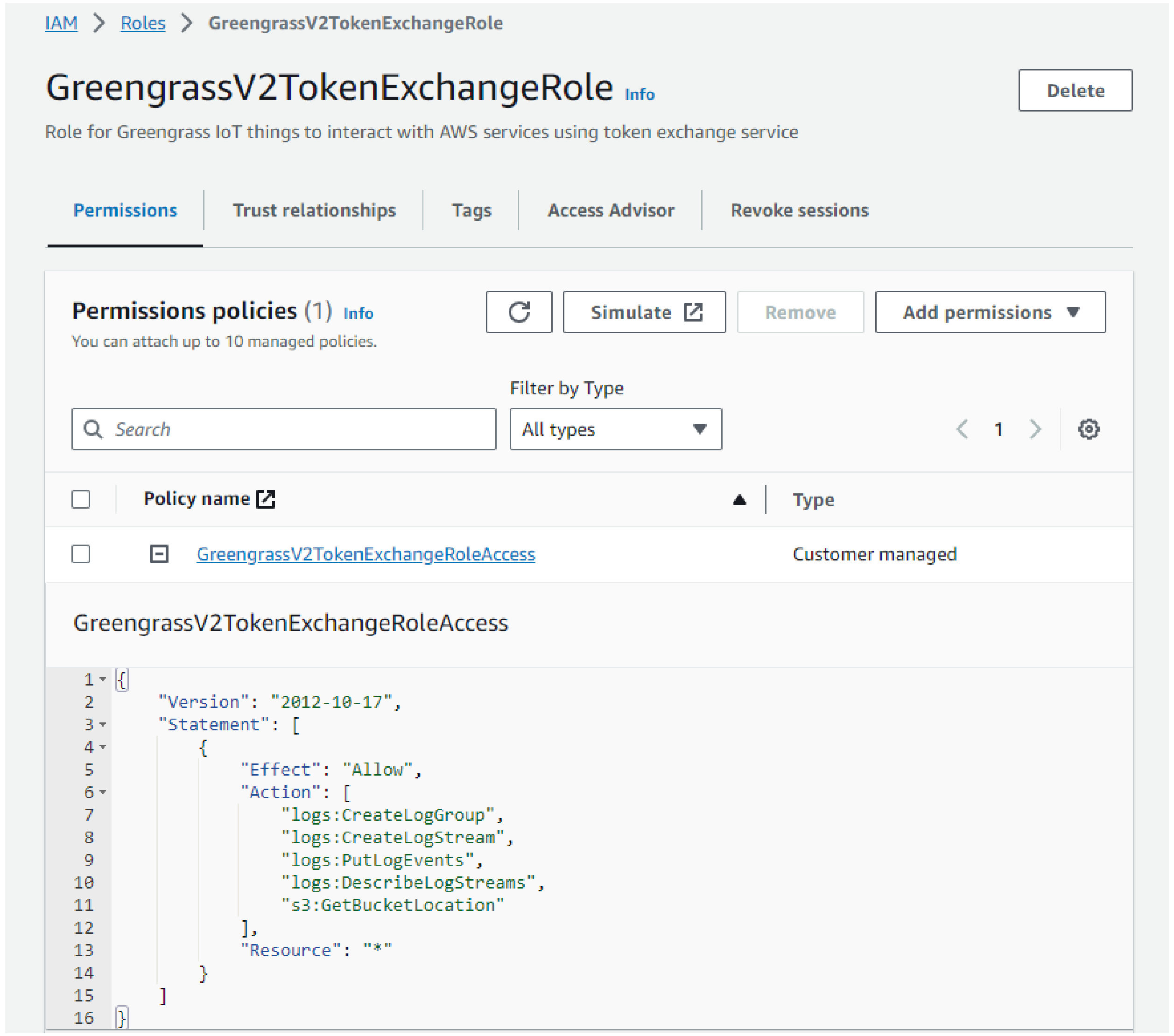
Rerun the Installer and Check If Core Device Is Healty


AWS IoT > Greengrass > Core devices
Info
Configure cloud discovery
Set up one core device
Search by core device name
1

Greengrass core devices

Name	Status	Status reported
CMPE583-GreengrassCore	Healthy	5 minutes ago

Step 4: Install the Greengrass Core Software



GreengrassV2TokenExchangeRole

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

Permissions policies (1)

You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type
GreengrassV2TokenExchangeRoleAccess	Customer managed

GreengrassV2TokenExchangeRoleAccess

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams",
                "s3:GetBucketLocation"
            ],
            "Resource": "*"
        }
    ]
}
```

- 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: Set Policies to Access Files in S3

Via AWS IoT Greengrass (Console)

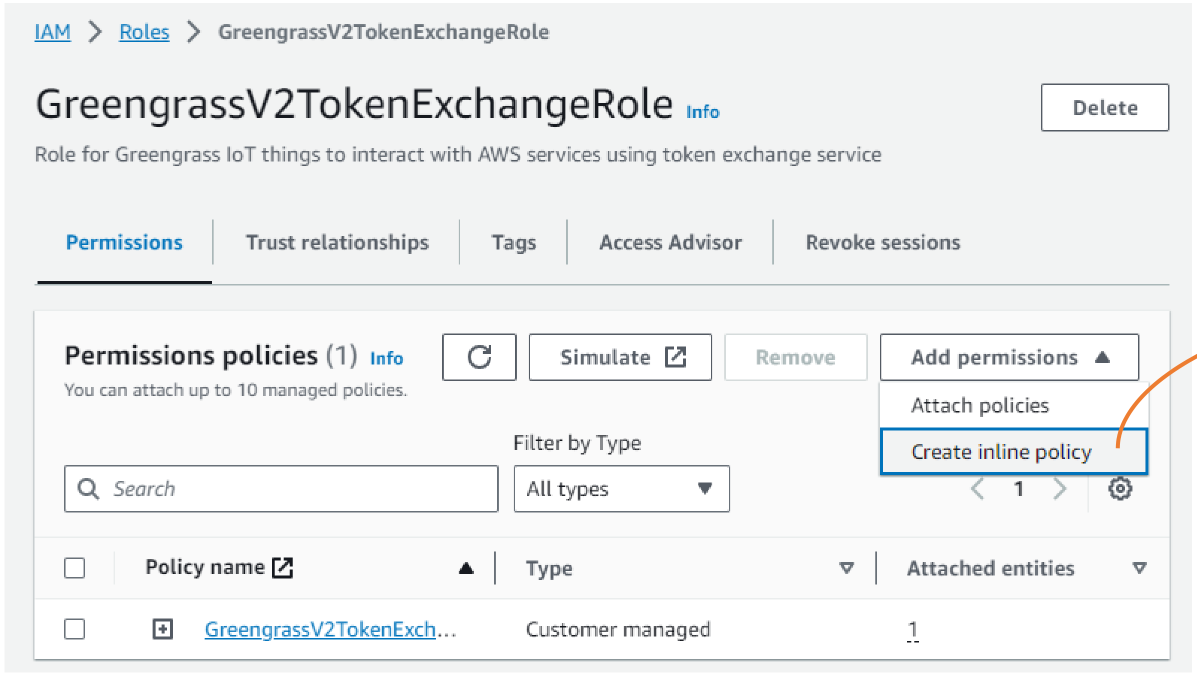
- Greengrass component artifacts should be stored in AWS S3 bucket.
- Therefore, you should allow the core device to access component artifacts in the S3 bucket.
  - Attach a policy similar to policy below to GreengrassV2TokenExchangeRole from AWS IAM console:

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

Step 1: Set Policies to Access Files in S3

Via AWS IoT Greengrass (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.



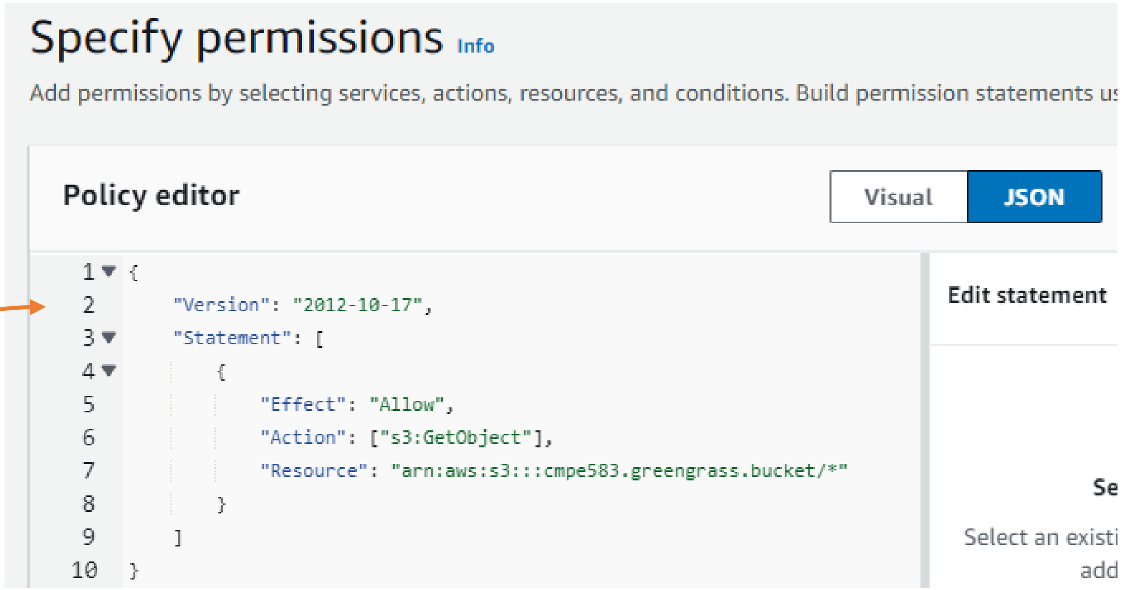
GreengrassV2TokenExchangeRole Info

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

Permissions policies (1) Info

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
GreengrassV2TokenExch...	Customer managed	1



Specify permissions Info

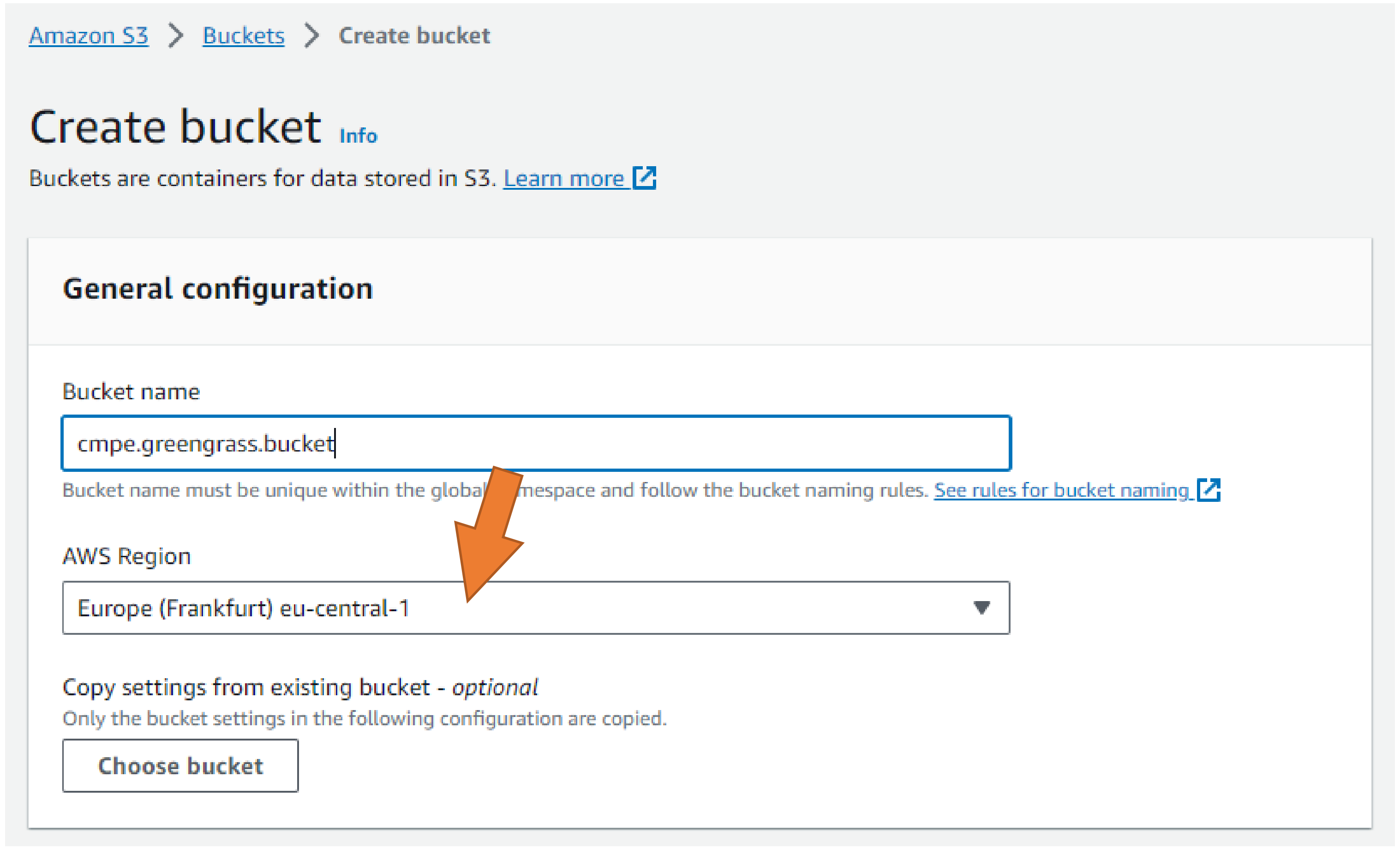
Add permissions by selecting services, actions, resources, and conditions. Build permission statements us

Policy editor

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

Step 1: Set Policies to Access Files in S3

Via AWS IoT Greengrass (Console)



Create bucket Info

Buckets are containers for data stored in S3. Learn more

General configuration

Bucket name

cmpe.greengrass.bucket

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

AWS Region

Europe (Frankfurt) eu-central-1

Copy settings from existing bucket - *optional*

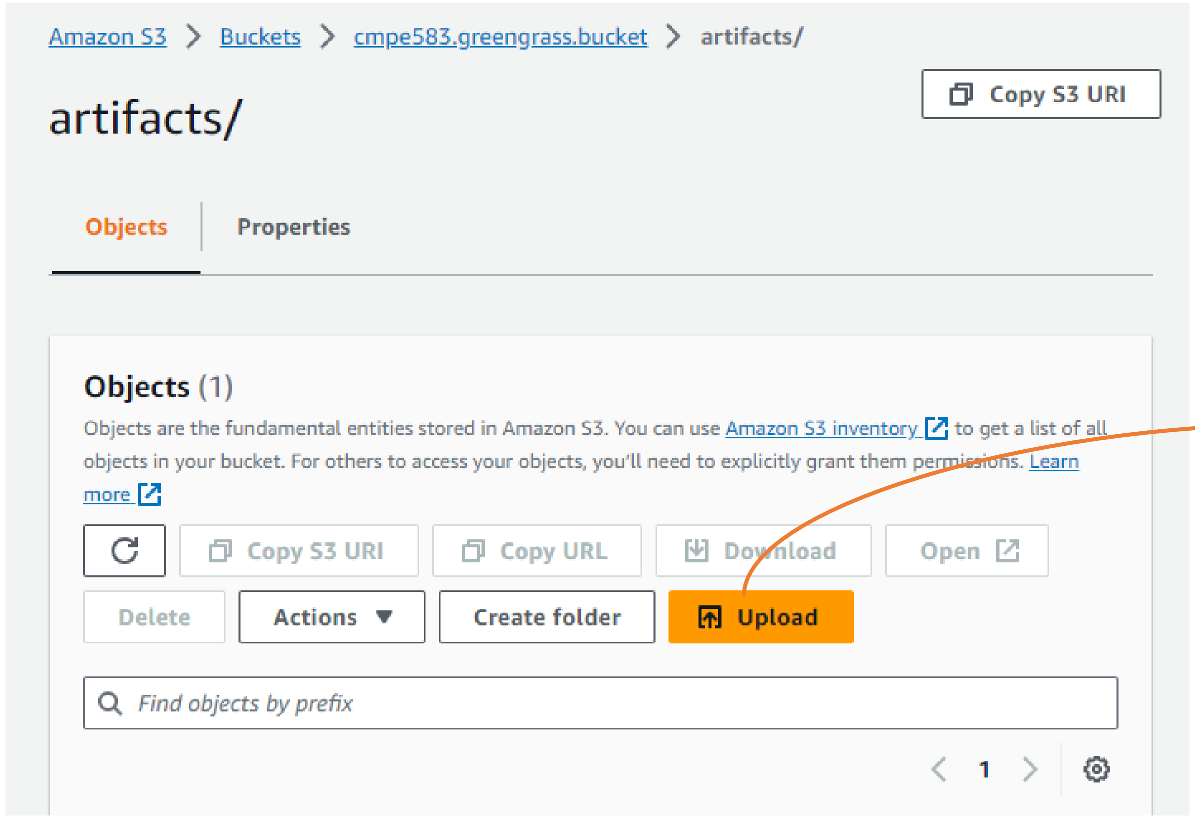
Only the bucket settings in the following configuration are copied.

- 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

Via AWS IoT Greengrass (Console)

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

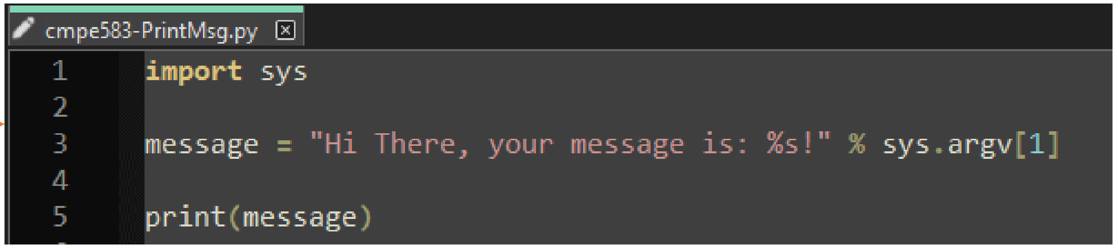


Amazon S3 > Buckets > cmpe583.greengrass.bucket > artifacts/

artifacts/

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

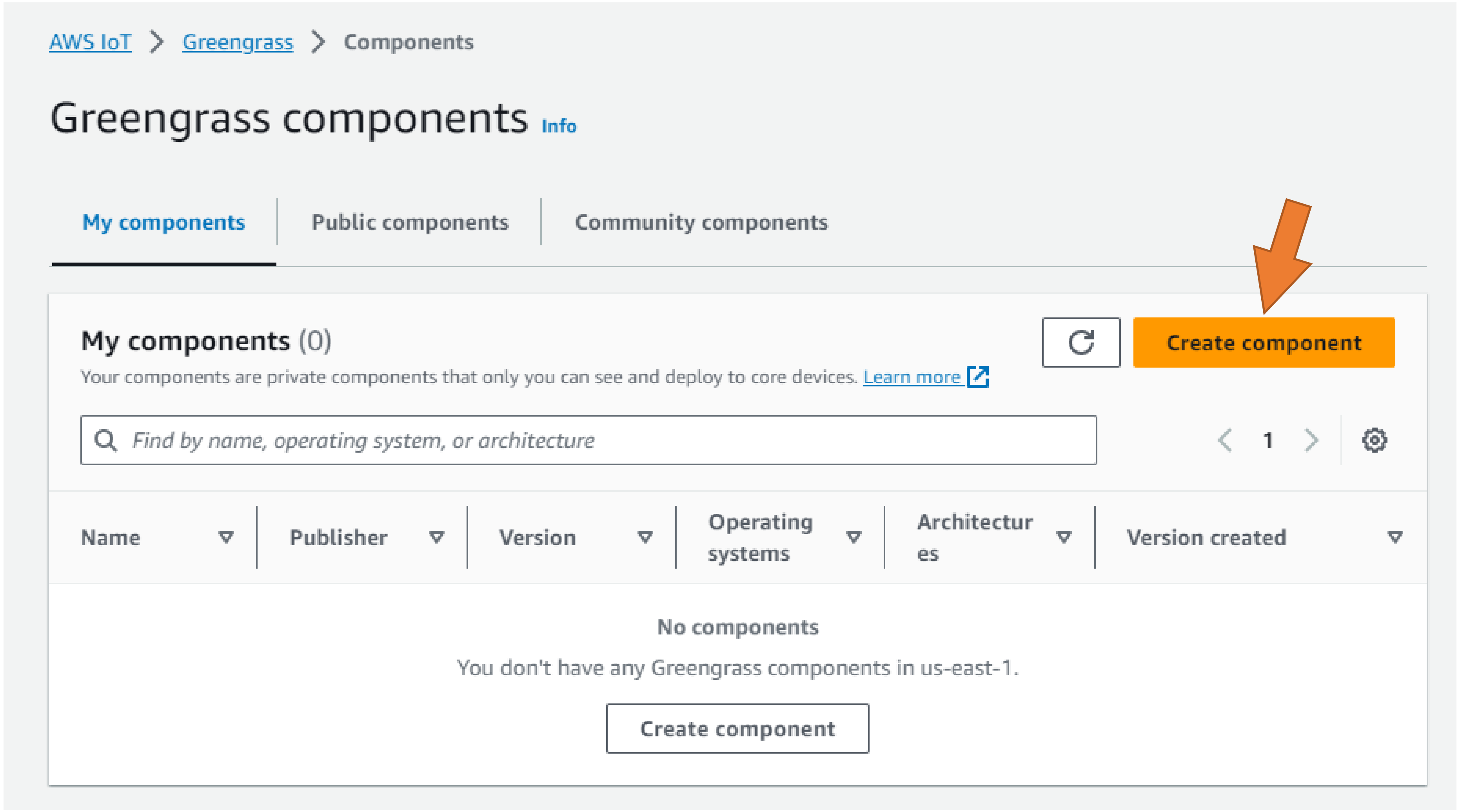


cmpe583-PrintMsg.py

```python
import sys

message = "Hi There, your message is: %s!" % sys.argv[1]

print(message)
```

Step 3: Create the Component

Via AWS IoT Greengrass (Console)



AWS IoT > Greengrass > Components

Greengrass components Info

My components | Public components | Community components

My components (0)

Your components are private components that only you can see and deploy to core devices. Learn more

Create component

Find by name, operating system, or architecture

Name	Publisher	Version	Operating systems	Architectures	Version created

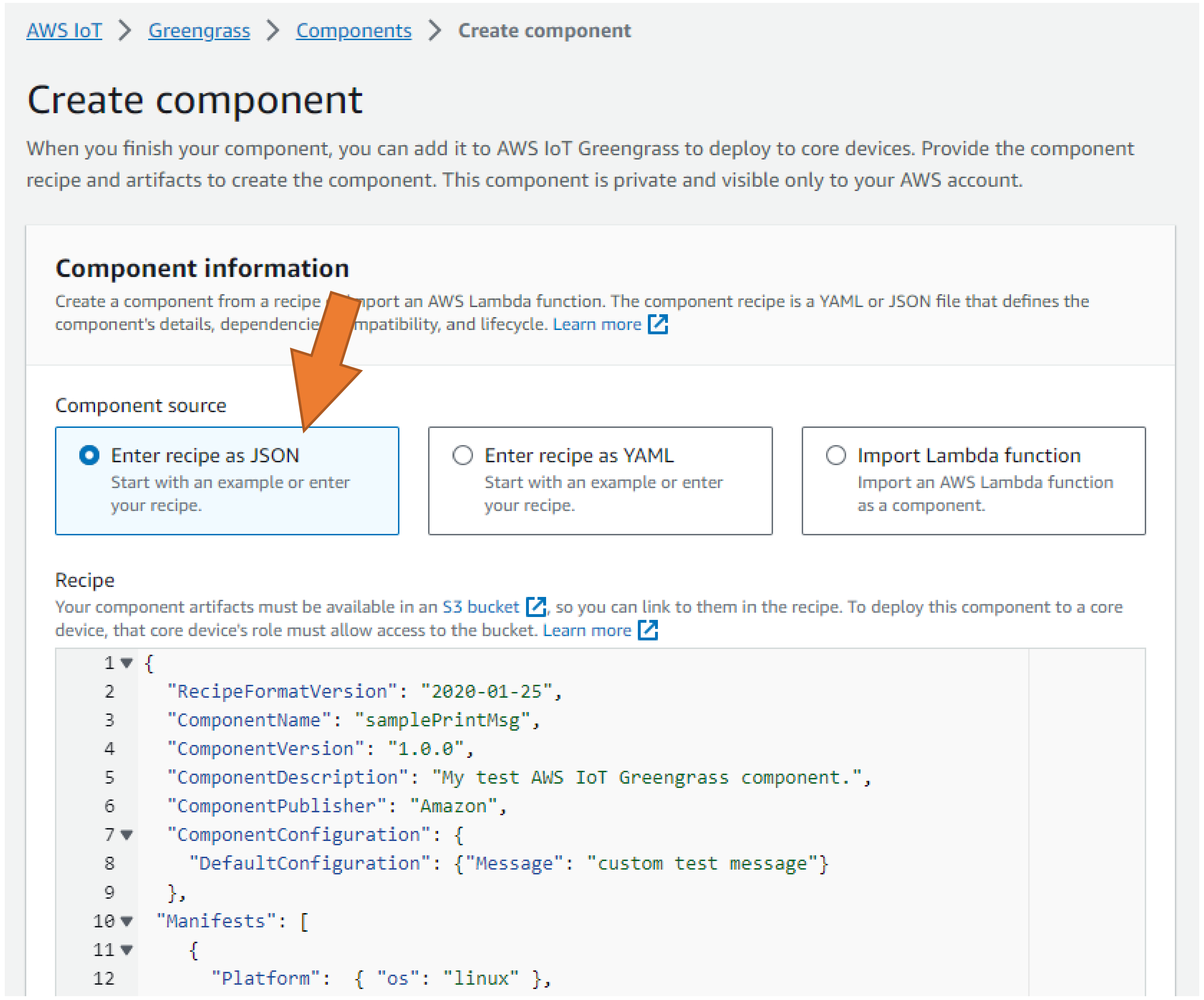
No components

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

Create component

Step 3: Create the Component

Define the Recipe



AWS IoT > Greengrass > Components > Create component

Create component

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

Component information

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

Component source

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

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

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

Recipe

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

```
{
  "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" },
```

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" } ]
   }
  ]
}
```

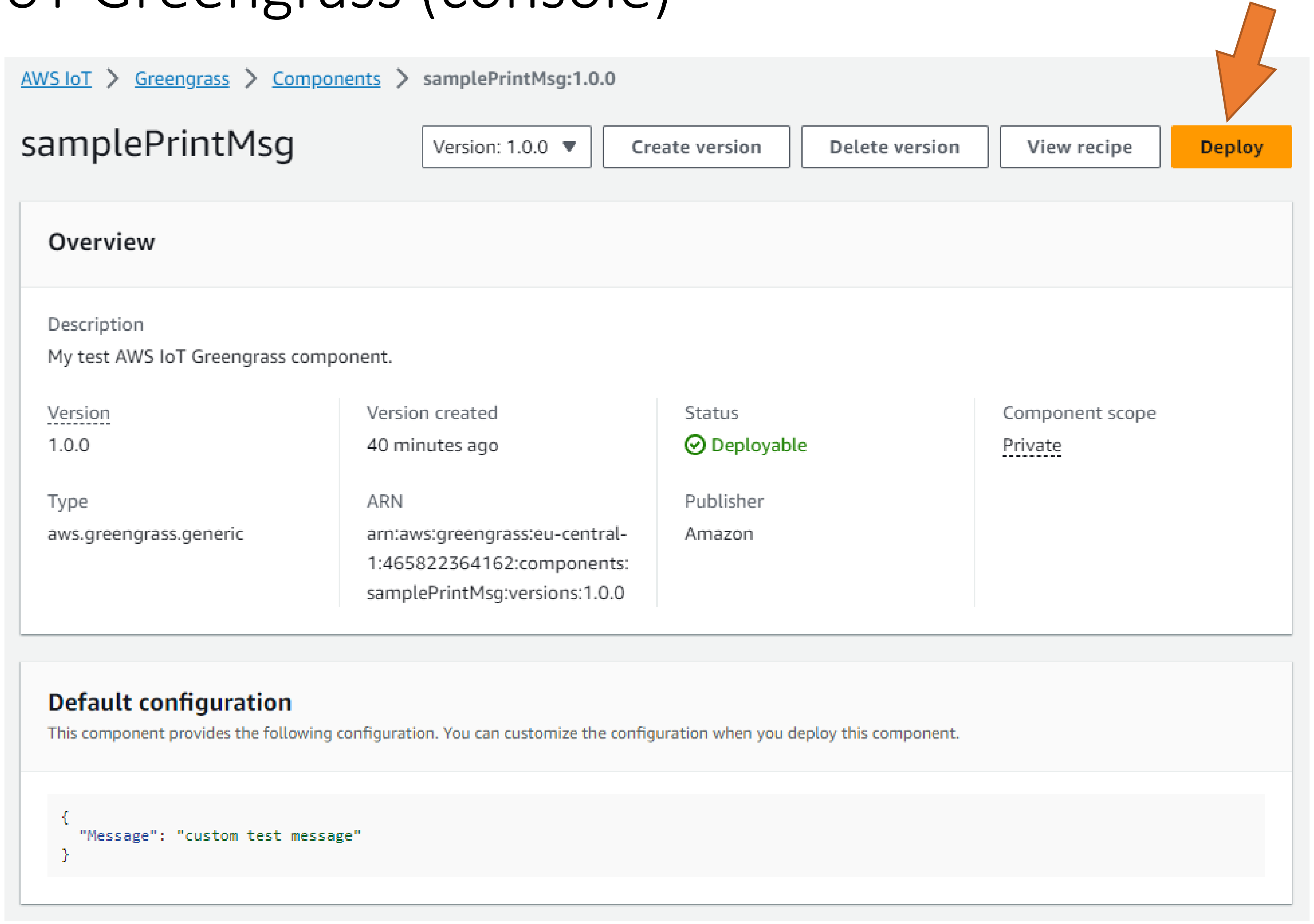
This test will be run on Linux device. So other OSs are ignored!

Define how to execute your artifact

Provide the artifact URI

Step 4: Deploy the Component

Via AWS IoT Greengrass (console)



Overview

Description			
My test AWS IoT Greengrass component.			
Version	Version created	Status	Component scope
1.0.0	40 minutes ago	✓ Deployable	Private
Type	ARN	Publisher	
aws.greengrass.generic	arn:aws:greengrass:eu-central-1:465822364162:components:samplePrintMsg:versions:1.0.0	Amazon	

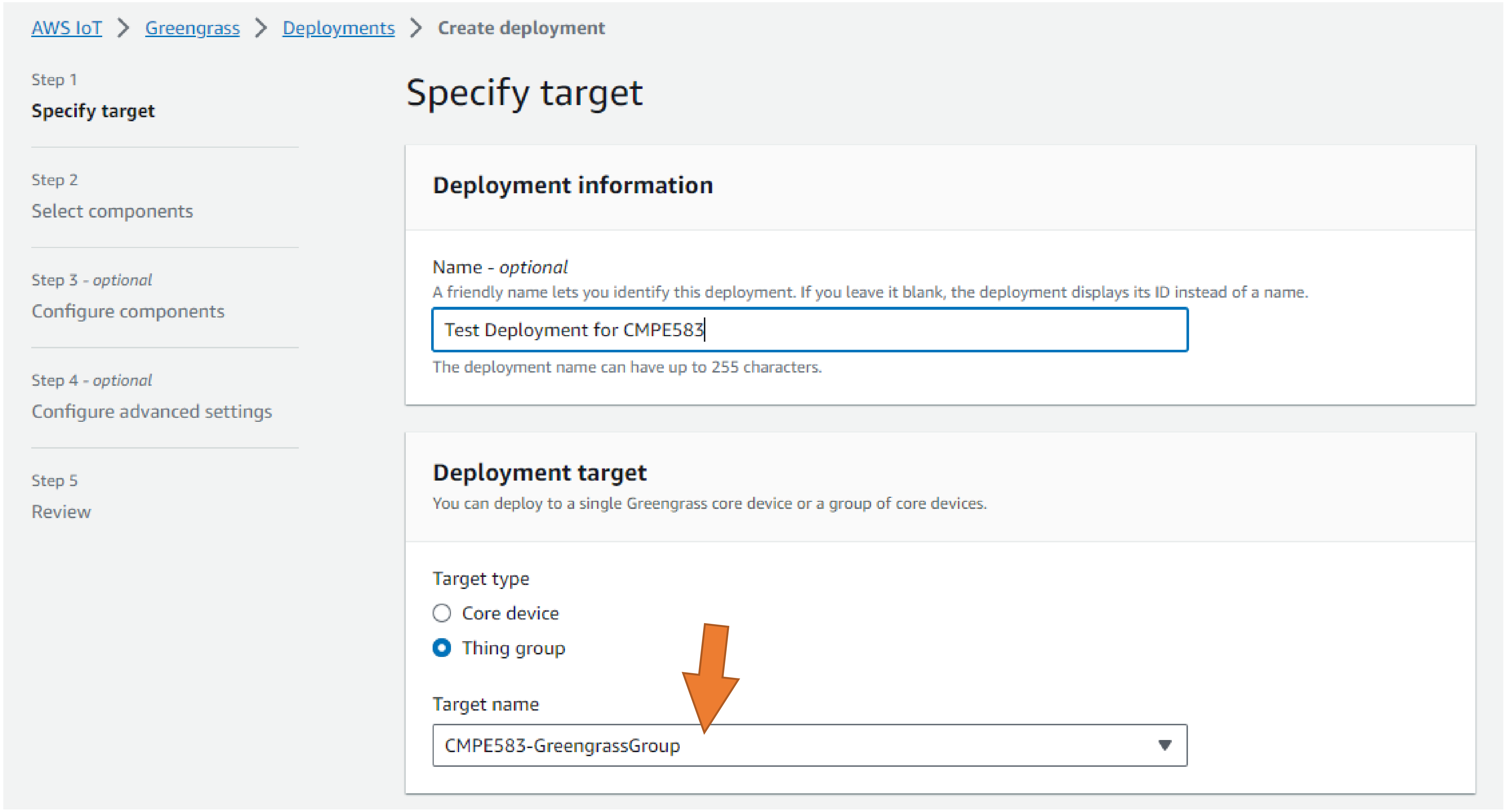
Default configuration

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

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

Step 4: Deploy the Component

Configure Deployment



AWS IoT > Greengrass > Deployments > Create deployment

Step 1
Specify target

Step 2
Select components

Step 3 - *optional*
Configure components

Step 4 - *optional*
Configure advanced settings

Step 5
Review

Specify target

Deployment information

Name - *optional*
A friendly name lets you identify this deployment. If you leave it blank, the deployment displays its ID instead of a name.
Test Deployment for CMPE583
The deployment name can have up to 255 characters.

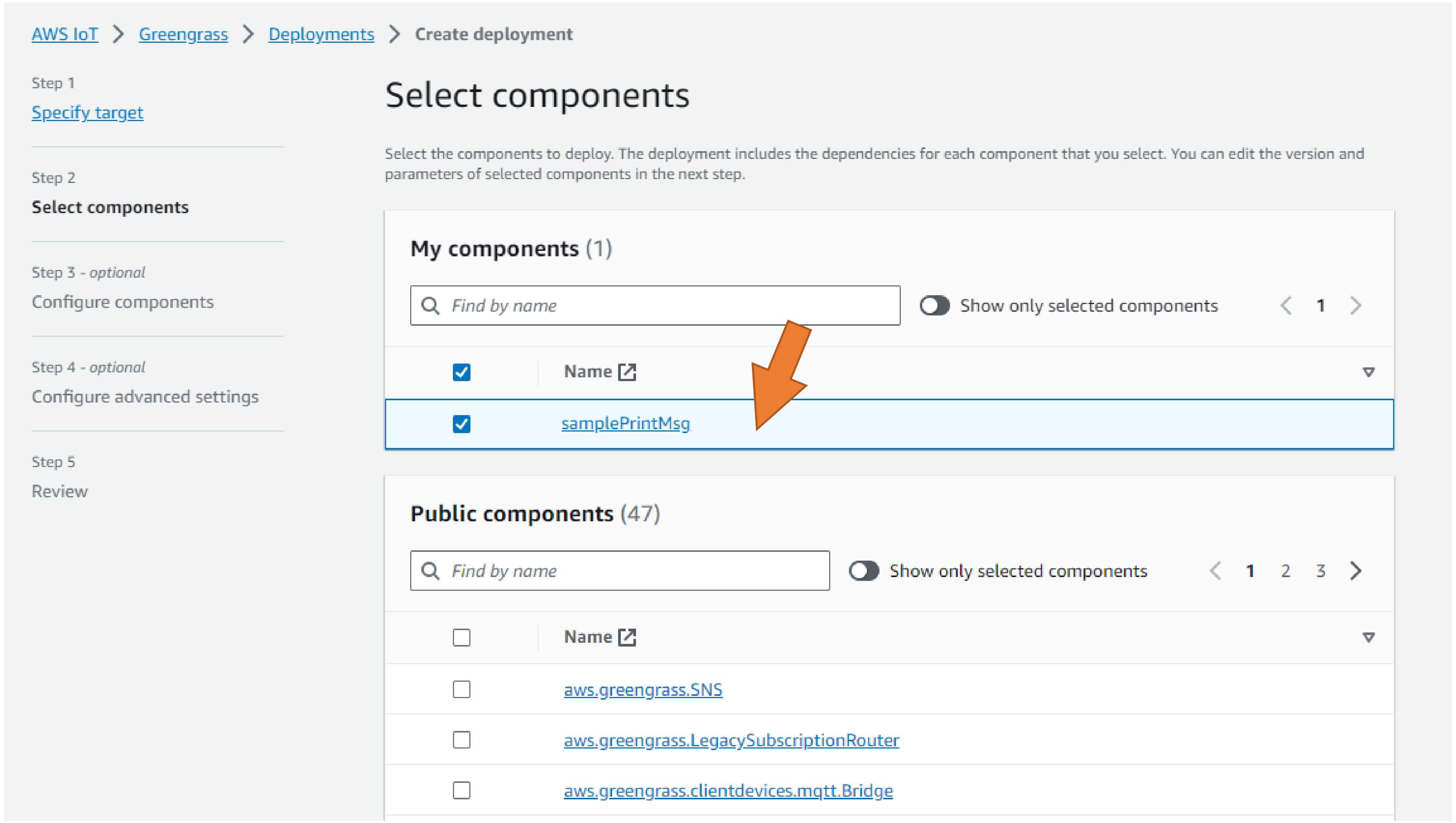
Deployment target
You can deploy to a single Greengrass core device or a group of core devices.

Target type
Core device [✓ Thing group]

Target name
CMPE583-GreengrassGroup

Step 4: Deploy the Component

Select the Component



AWS IoT > Greengrass > Deployments > Create deployment

Step 1
Specify target

Step 2
Select components

Step 3 - *optional*
Configure components

Step 4 - *optional*
Configure advanced settings

Step 5
Review

Select components

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

My components (1)

Find by name | Show only selected components | < 1 >

	Name
✓	samplePrintMsg

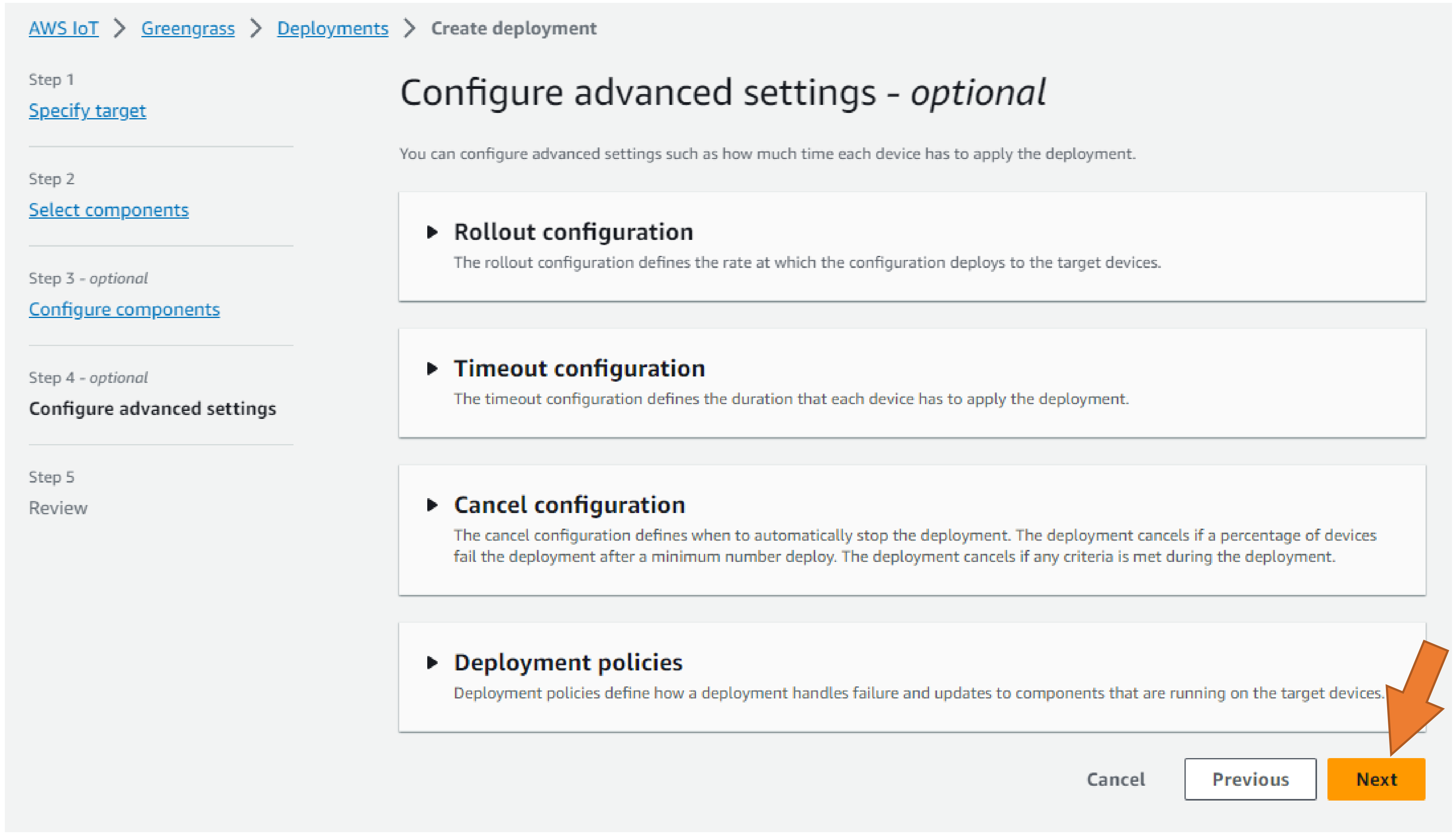
Public components (47)

Find by name | Show only selected components | < 1 2 3 >

	Name
☐	aws.greengrass.SNS
☐	aws.greengrass.LegacySubscriptionRouter
☐	aws.greengrass.clientdevices.mqtt.Bridge

Step 4: Deploy the Component

Configure Advanced Settings



AWS IoT > Greengrass > Deployments > Create deployment

Step 1
Specify target

Step 2
Select components

Step 3 - *optional*
Configure components

Step 4 - *optional*
Configure advanced settings

Step 5
Review

Configure advanced settings - *optional*

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

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

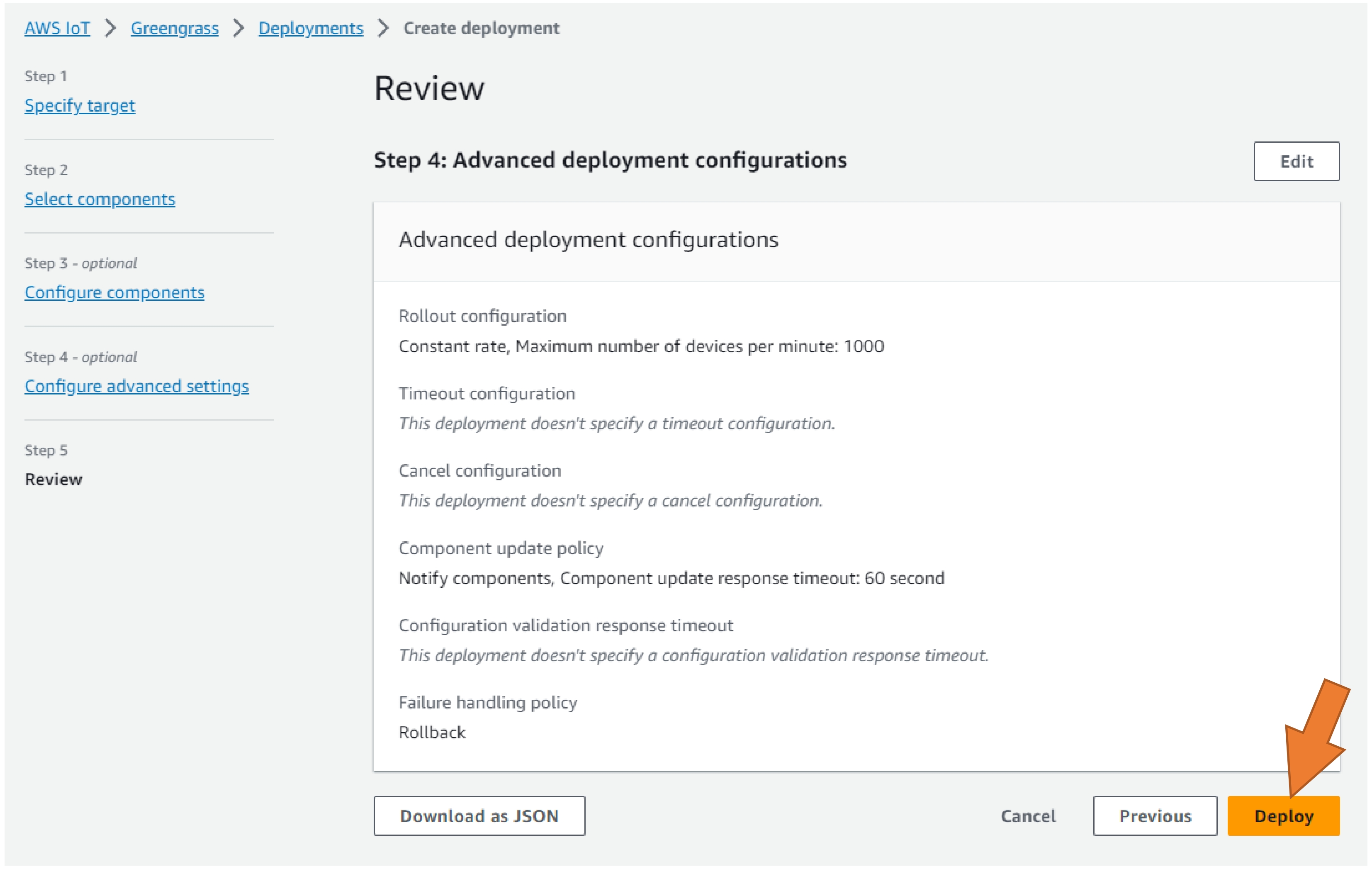
▶ **Timeout configuration**
The timeout configuration defines the duration that each device has to apply the deployment.

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

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

Step 4: Deploy the Component

Review & Deploy



AWS IoT > Greengrass > Deployments > Create deployment

Step 1
Specify target

Step 2
Select components

Step 3 - *optional*
Configure components

Step 4 - *optional*
Configure advanced settings

Step 5
Review

Review

Step 4: Advanced deployment configurations

Advanced deployment configurations

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

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

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

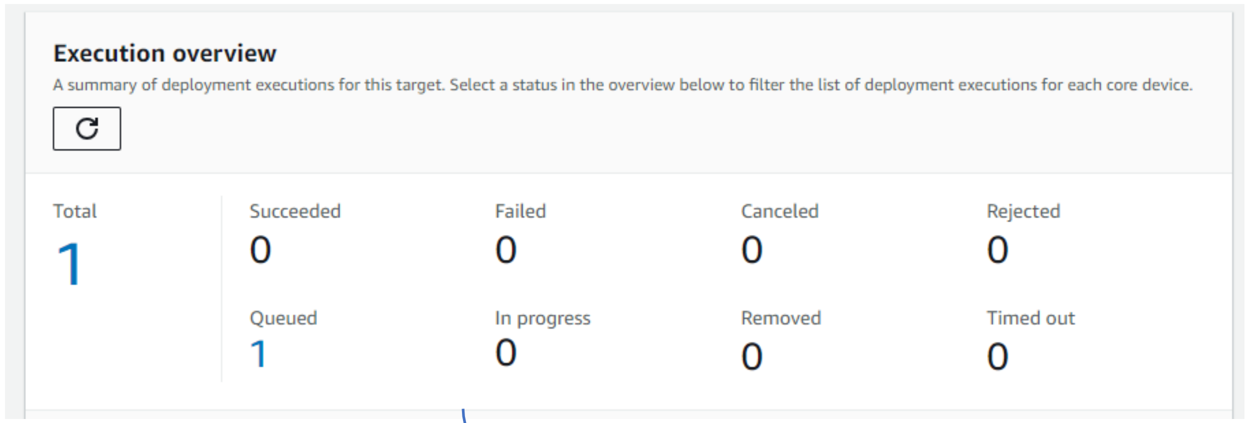
Component update policy
Notify components, Component update response timeout: 60 second

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

Failure handling policy
Rollback

Step 5: Verify the Component

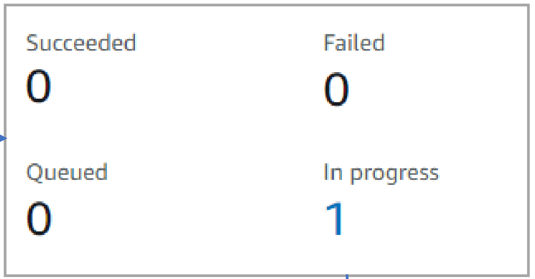
Check AWS IoT Greengrass Console



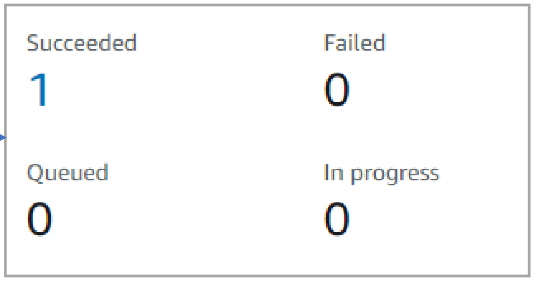
Execution overview

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

Total	Succeeded	Failed	Canceled	Rejected
1	0	0	0	0
	Queued	In progress	Removed	Timed out
	1	0	0	0



Succeeded	Failed
0	0
Queued	In progress
0	1

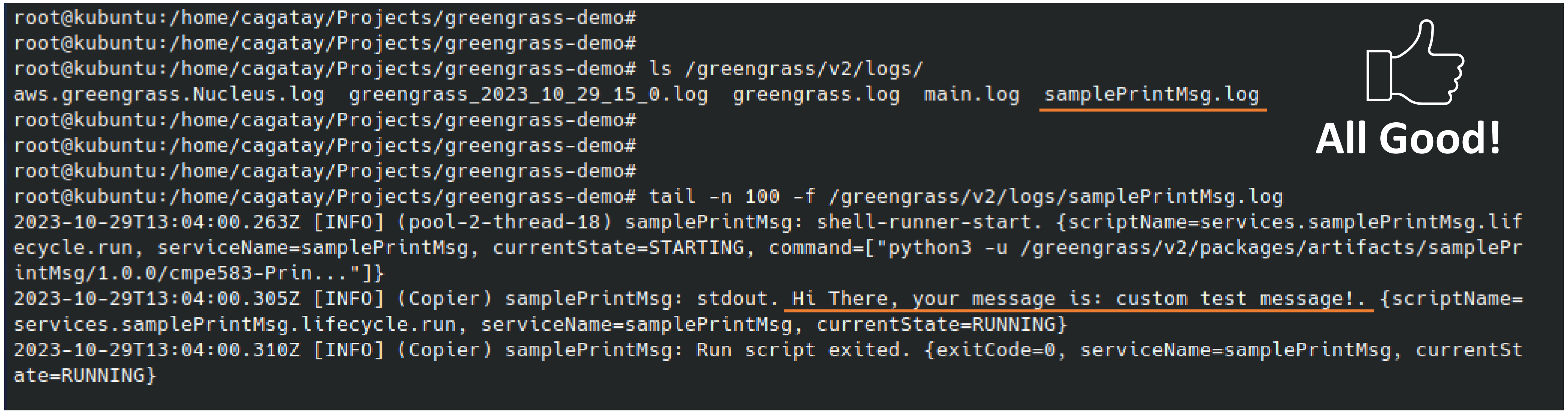


Succeeded	Failed
1	0
Queued	In progress
0	0

Step 5: Verify the Component

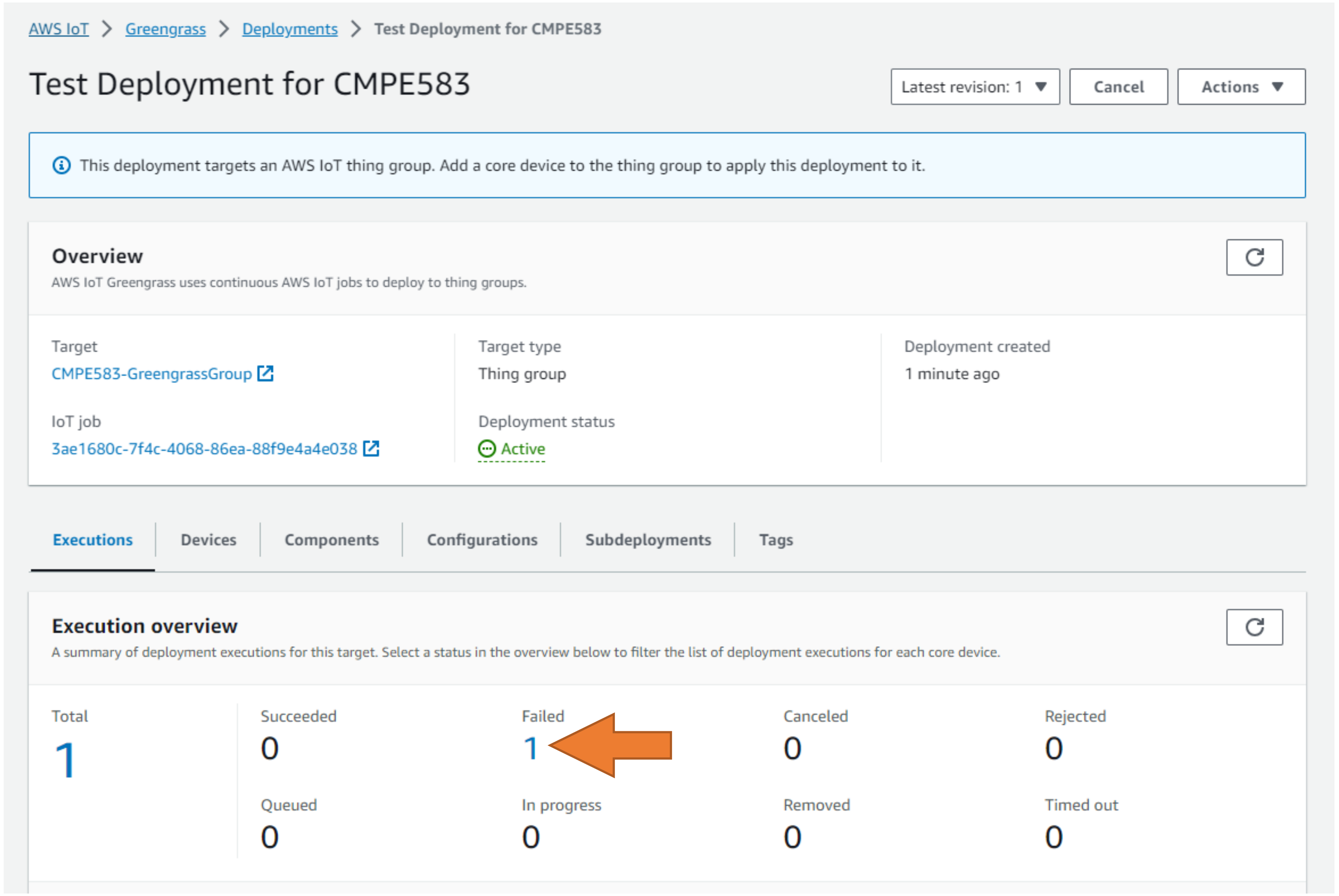
Check Log Files

➢ 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}
```

Troubleshooting



AWS IoT > Greengrass > Deployments > Test Deployment for CMPE583

Test Deployment for CMPE583

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

Overview

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

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

Execution overview

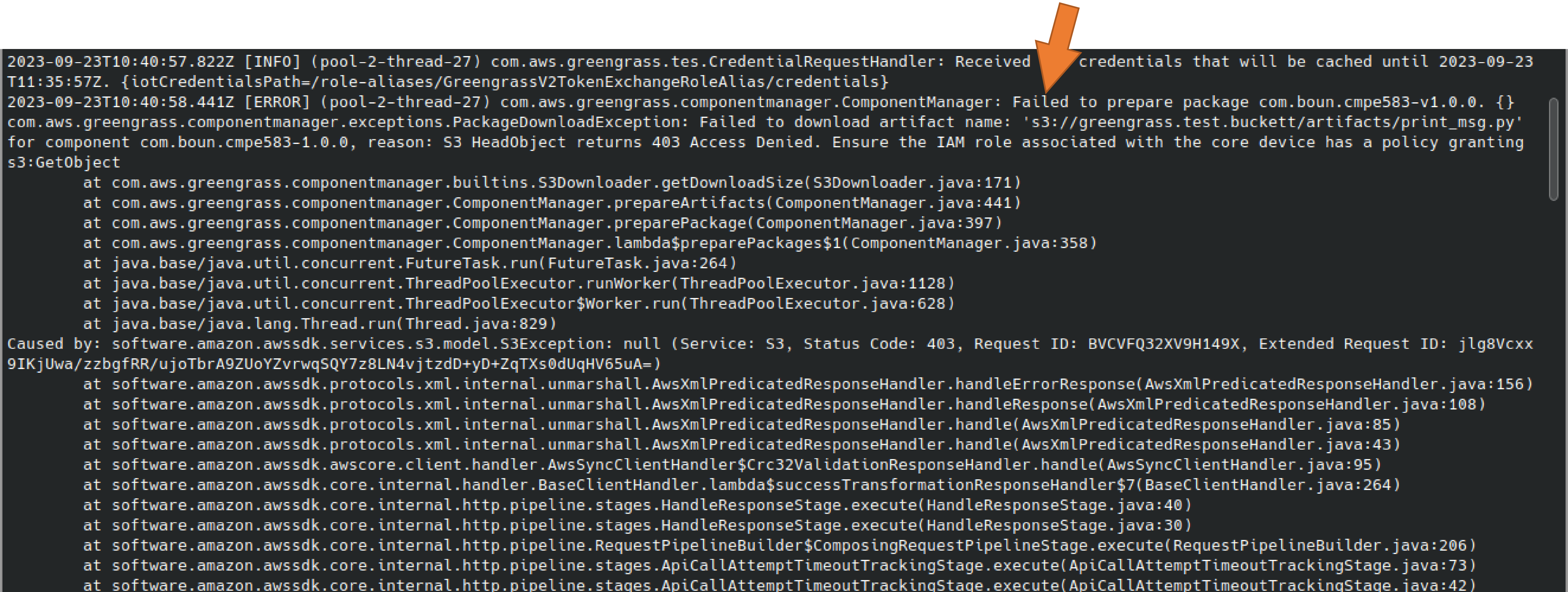
A summary of deployment executions for this target. Select a status in the overview below to filter the list of deployment executions for each core device.

Total	Succeeded	Failed	Canceled	Rejected
1	0	1	0	0
	Queued	In progress	Removed	Timed out
	0	0	0	0

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

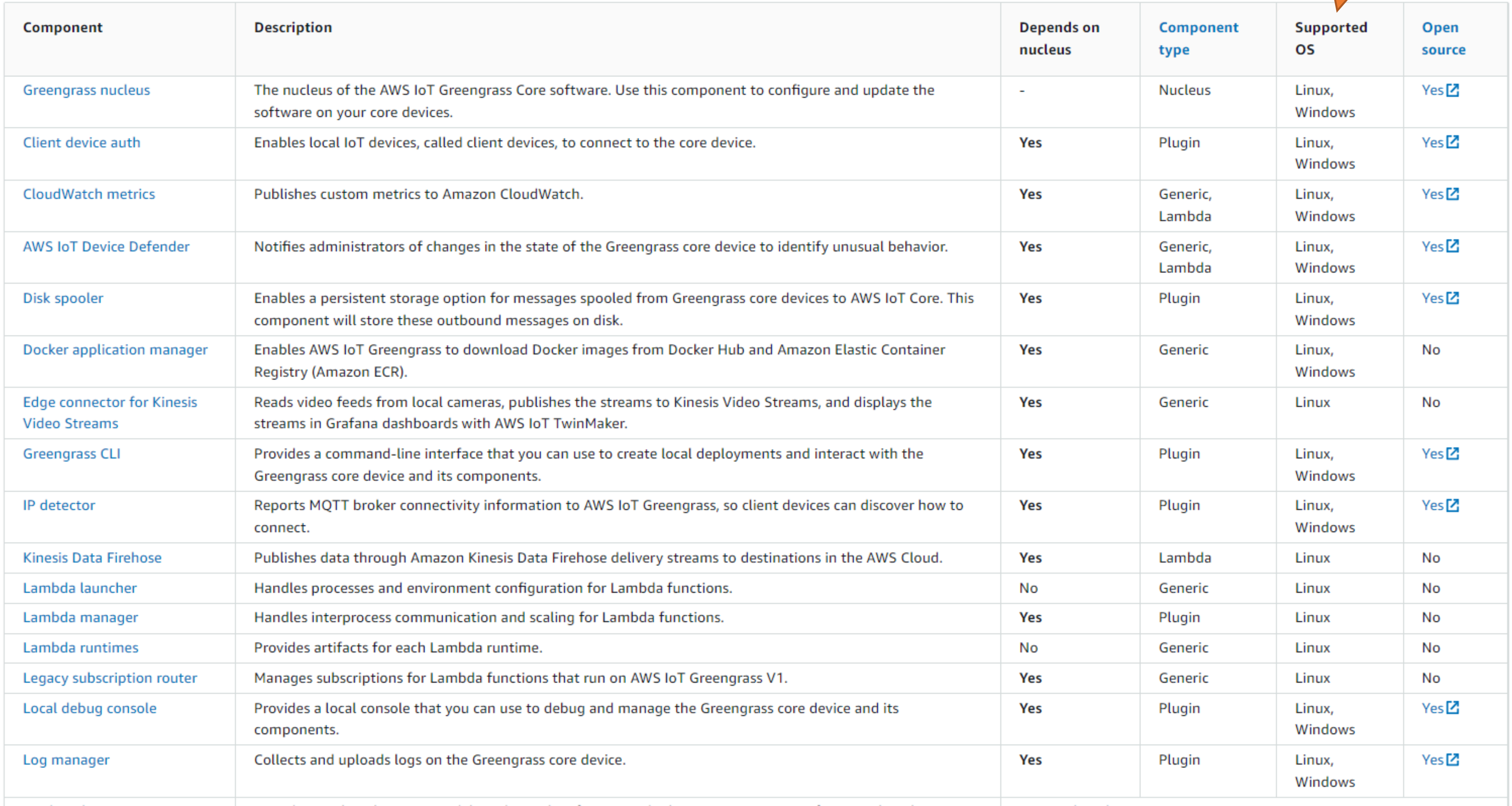
AWS Provided Components

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

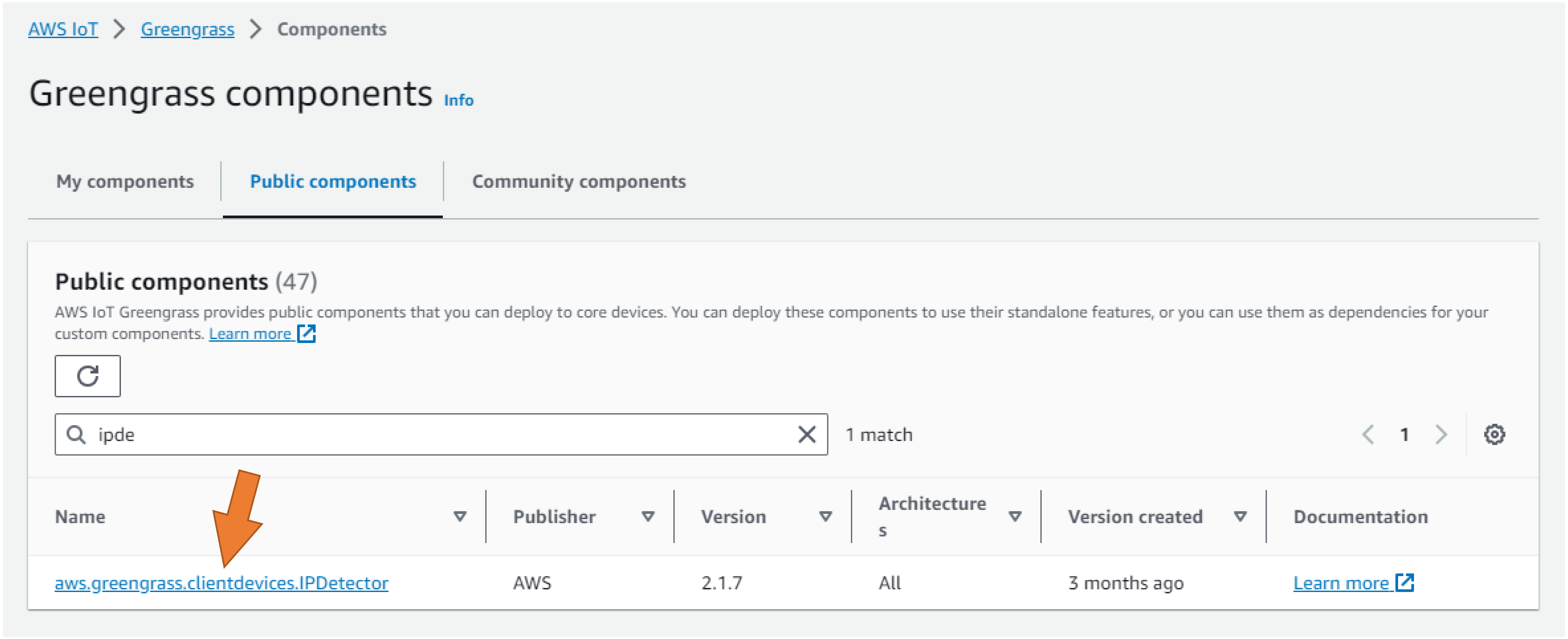
OS Support and More



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



AWS IoT > Greengrass > Components

Greengrass components Info

My components | Public components | Community components

Public components (47)

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

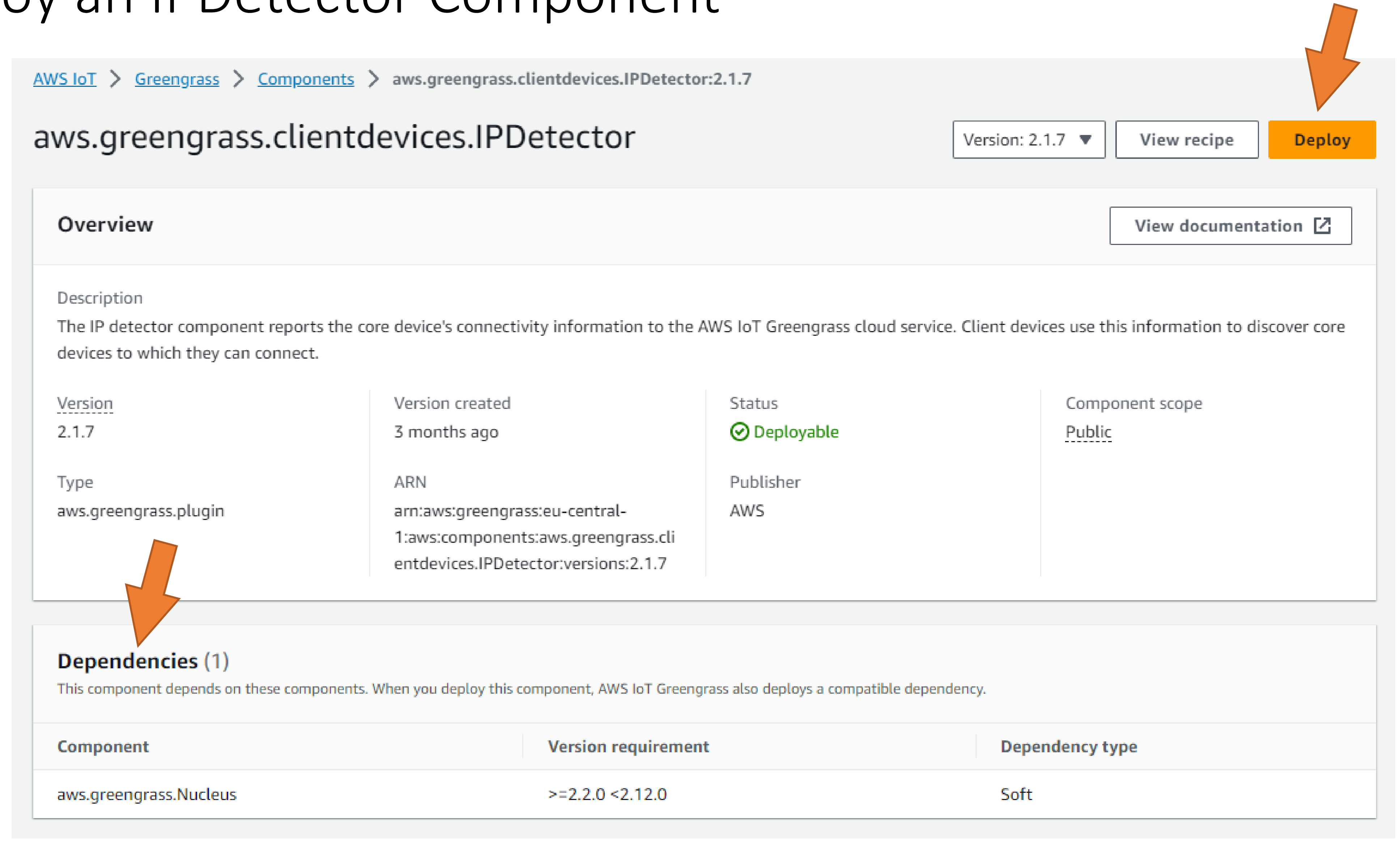
ipde

1 match

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

Collecting IP Addresses of Core Devices

Deploy an IPDetector Component



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

aws.greengrass.clientdevices.IPDetector

Overview

Description

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

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

Dependencies (1)

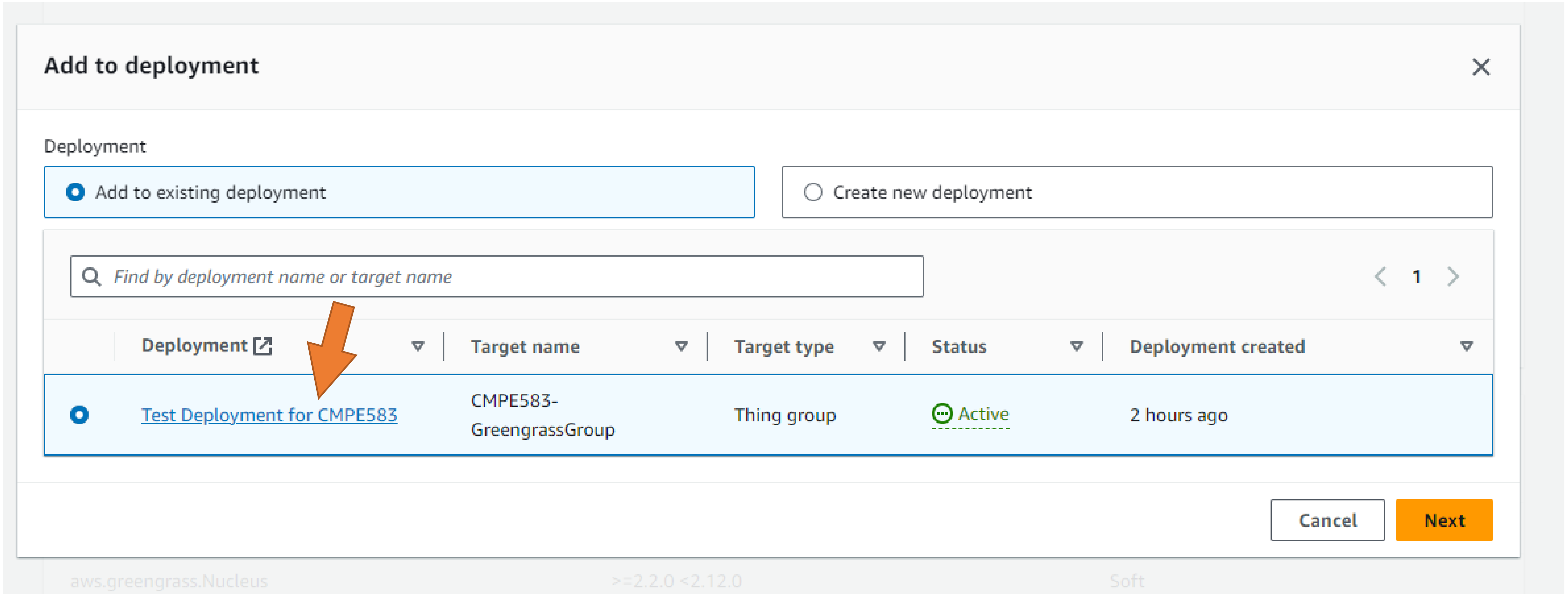
This component depends on these components. When you deploy this component, AWS IoT Greengrass also deploys a compatible dependency.

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

Collecting IP Addresses of Core Devices

Add Public Component to Existing Deployment

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



Add to deployment

Deployment

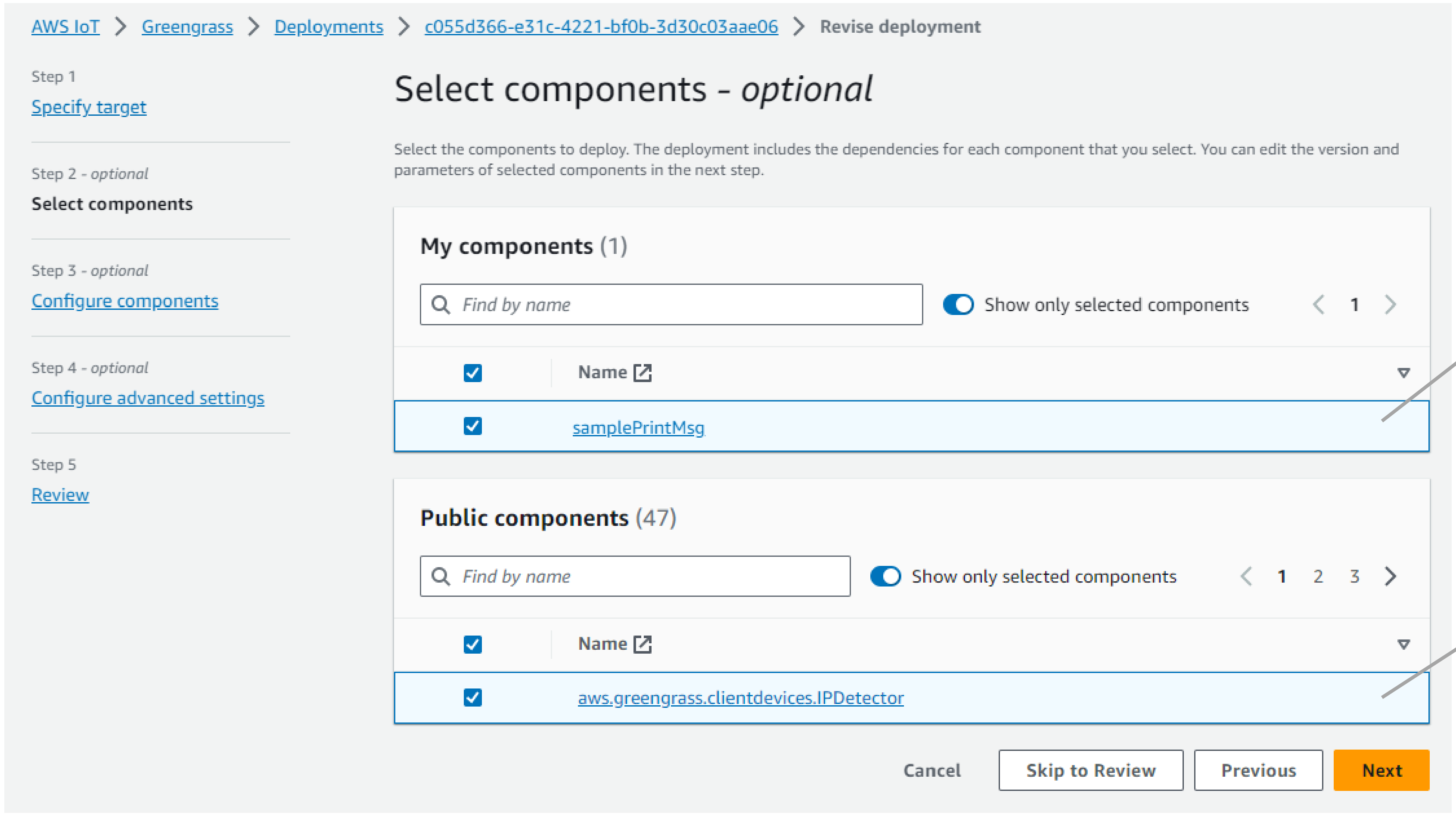
[✓ Add to existing deployment] Create new deployment

Find by deployment name or target name

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

Collecting IP Addresses of Core Devices

Select All Components You Want to Deploy



AWS IoT > Greengrass > Deployments > c055d366-e31c-4221-bf0b-3d30c03aae06 > Revise deployment

Step 1
Specify target

Step 2 - *optional*
Select components

Step 3 - *optional*
Configure components

Step 4 - *optional*
Configure advanced settings

Step 5
Review

Select components - *optional*

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

My components (1)

Find by name | Show only selected components (on) | 1

✓	Name
✓	samplePrintMsg

Public components (47)

Find by name | Show only selected components (on) | 1 2 3

✓	Name
✓	aws.greengrass.clientdevices.IPDetector

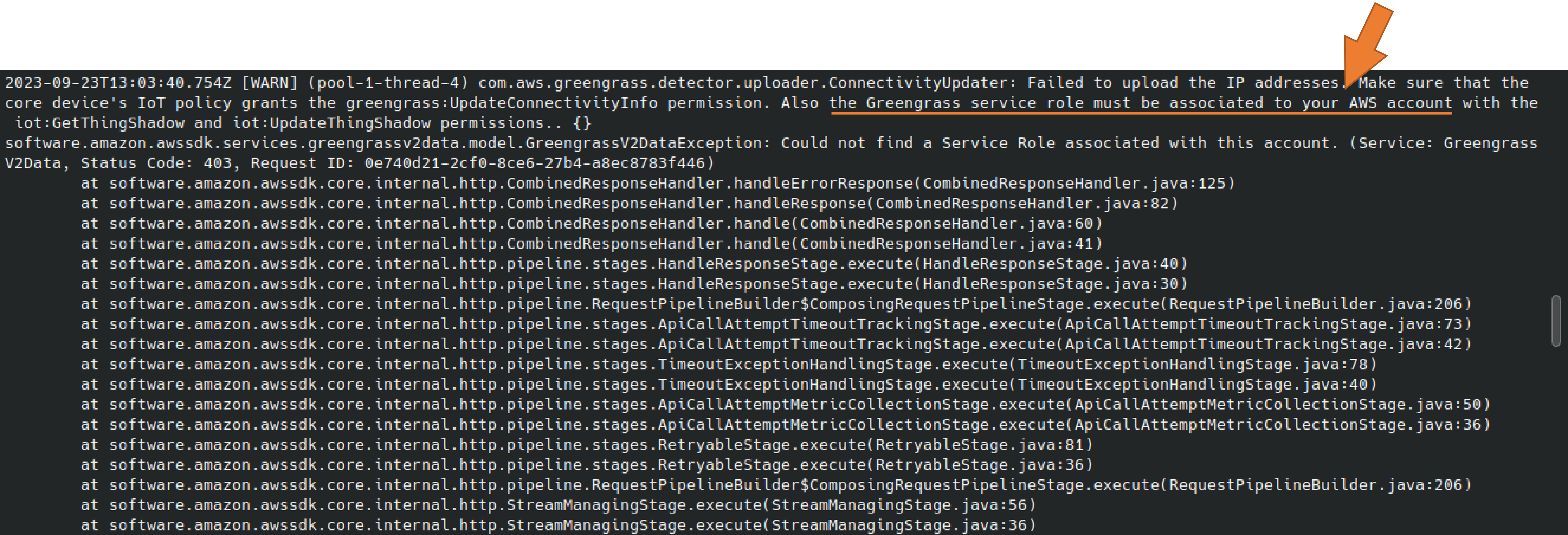
Custom component

AWS Provided component

Collecting IP Addresses of Core Devices

Troubleshooting

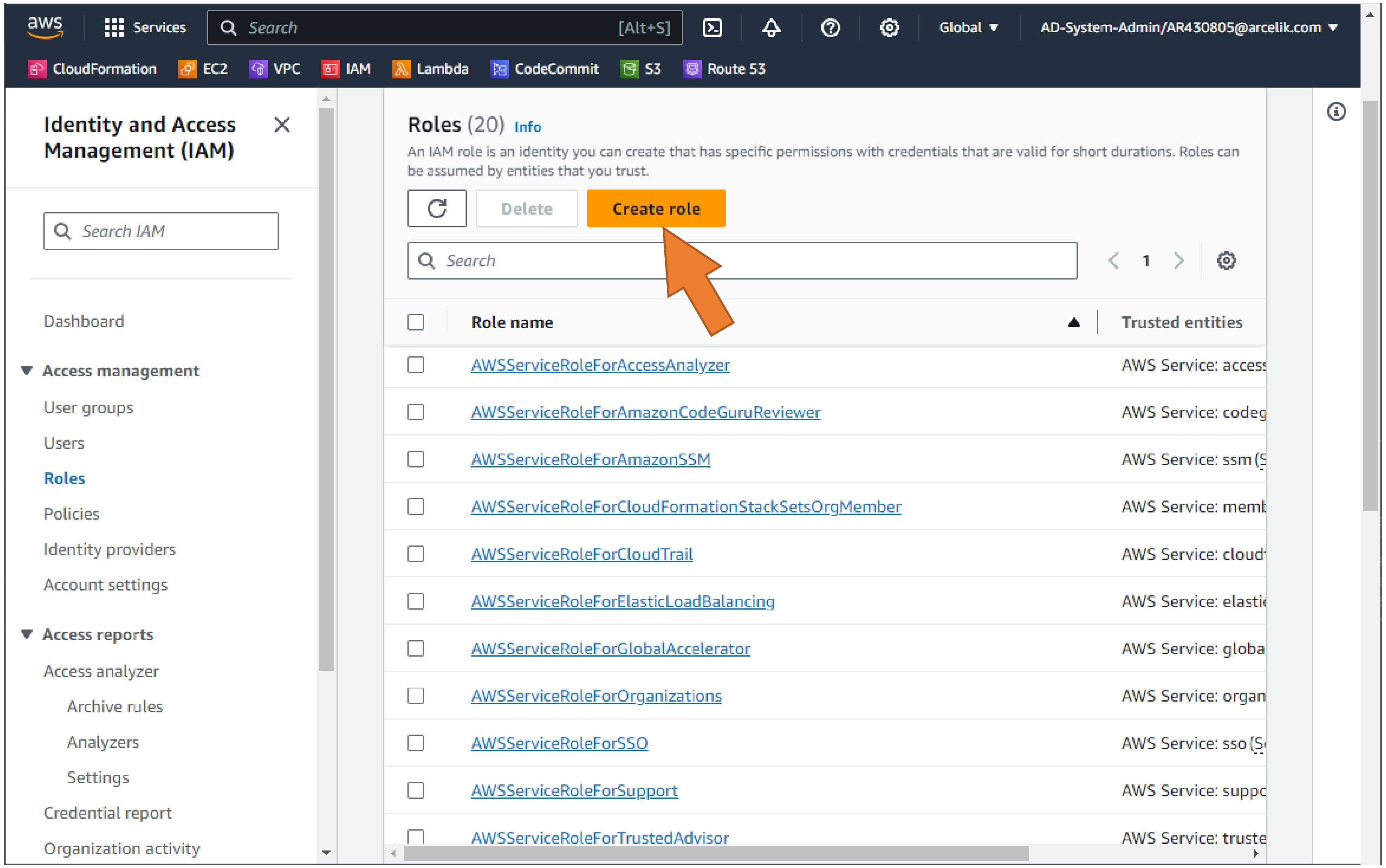
- Check greengrass.log file if the deployed job runs without error.
- You will see an error if the Greengrass service role is not associated to your AWS account.
- Greengrass needs permission to access the AWS services on your behalf.



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

Troubleshooting

Create an IAM Role



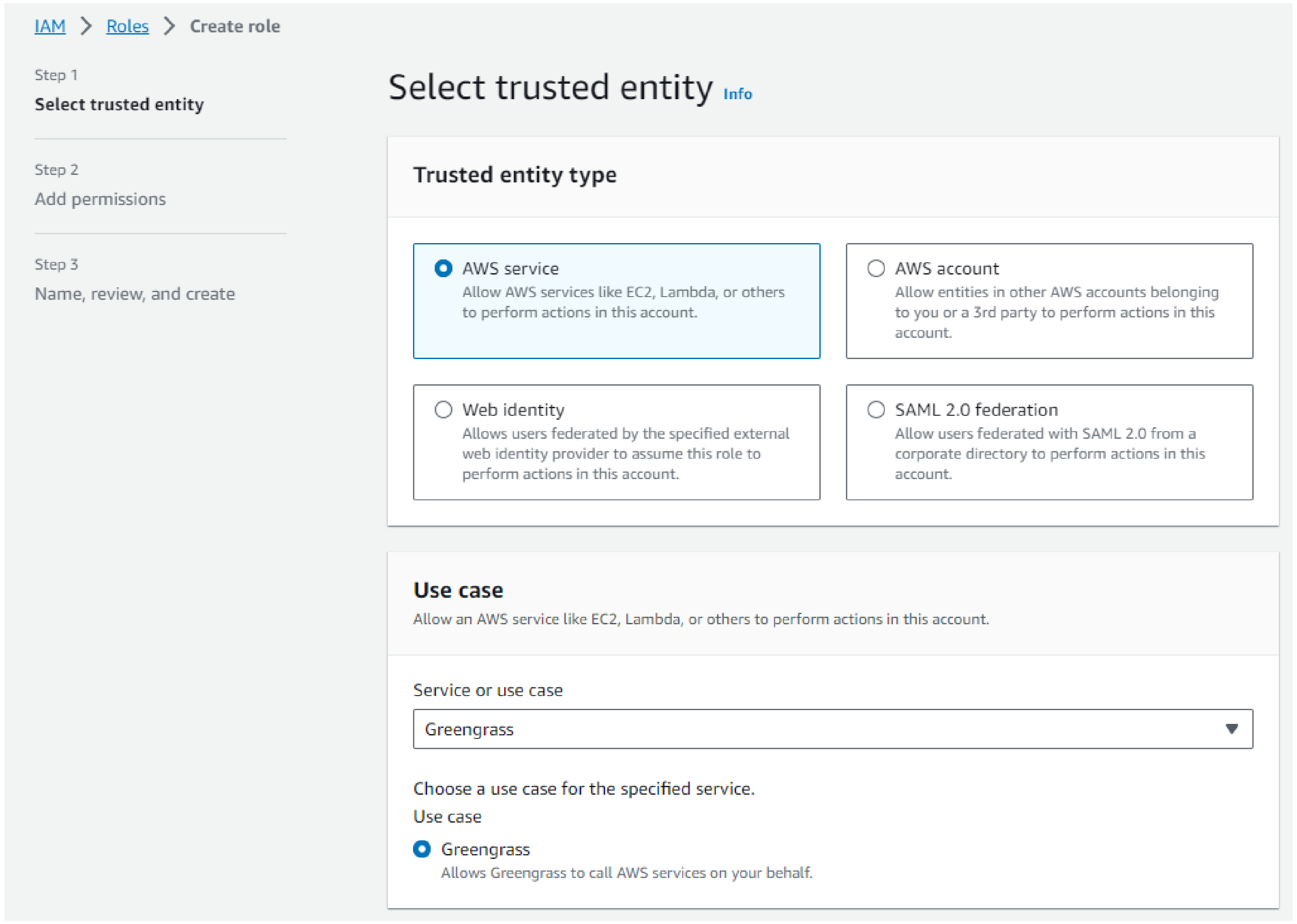
Roles (20)

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities
AWSServiceRoleForAccessAnalyzer	AWS Service: access
AWSServiceRoleForAmazonCodeGuruReviewer	AWS Service: codeg
AWSServiceRoleForAmazonSSM	AWS Service: ssm (S
AWSServiceRoleForCloudFormationStackSetsOrgMember	AWS Service: memb
AWSServiceRoleForCloudTrail	AWS Service: cloud
AWSServiceRoleForElasticLoadBalancing	AWS Service: elasti
AWSServiceRoleForGlobalAccelerator	AWS Service: globa
AWSServiceRoleForOrganizations	AWS Service: organ
AWSServiceRoleForSSO	AWS Service: sso (S
AWSServiceRoleForSupport	AWS Service: suppo
AWSServiceRoleForTrustedAdvisor	AWS Service: truste

Troubleshooting

Create an IAM Role



IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity Info

Trusted entity type

- [✓ AWS service] Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account — Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity — Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- SAML 2.0 federation — Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

Use case

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

Service or use case: Greengrass

Choose a use case for the specified service.

Use case

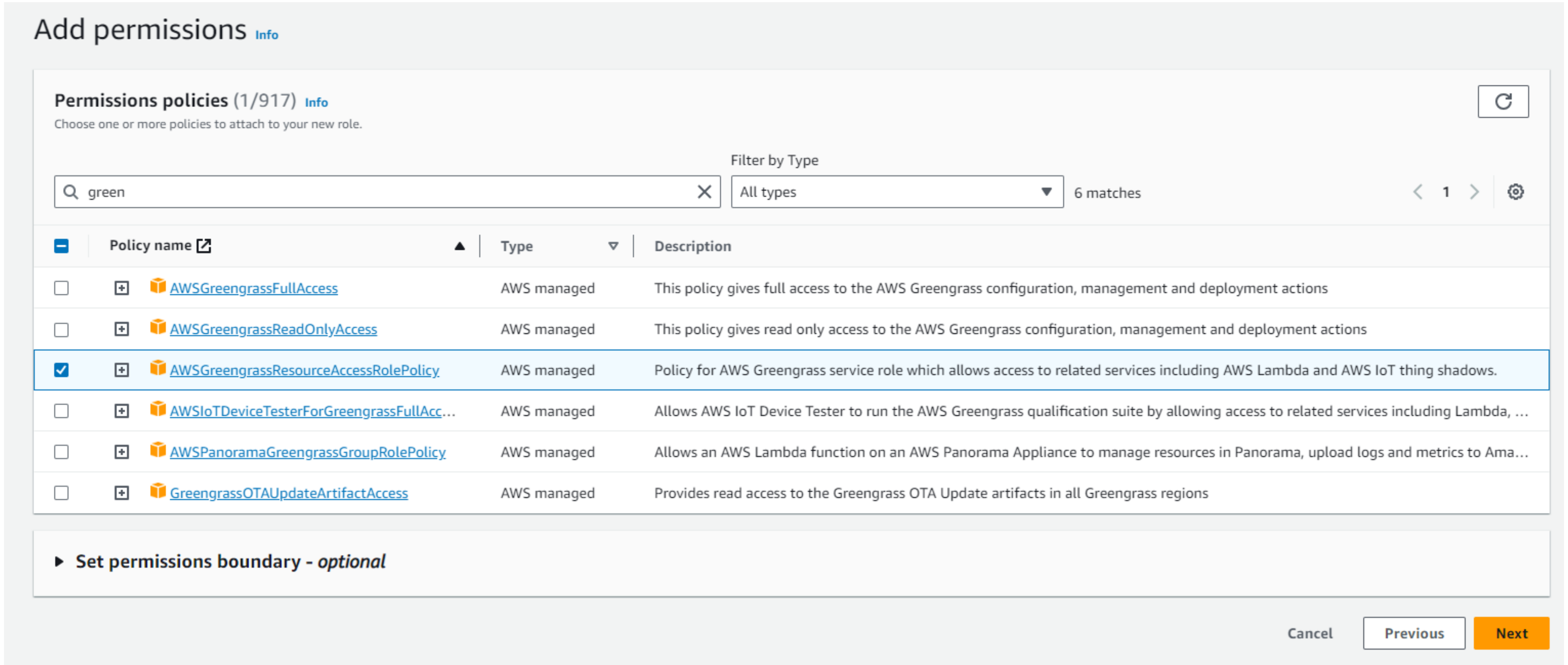
- [✓ Greengrass] Allows Greengrass to call AWS services on your behalf.

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

Troubleshooting

Add Permission to Created Role

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



Add permissions Info

Permissions policies (1/917) Info

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

Filter by Type

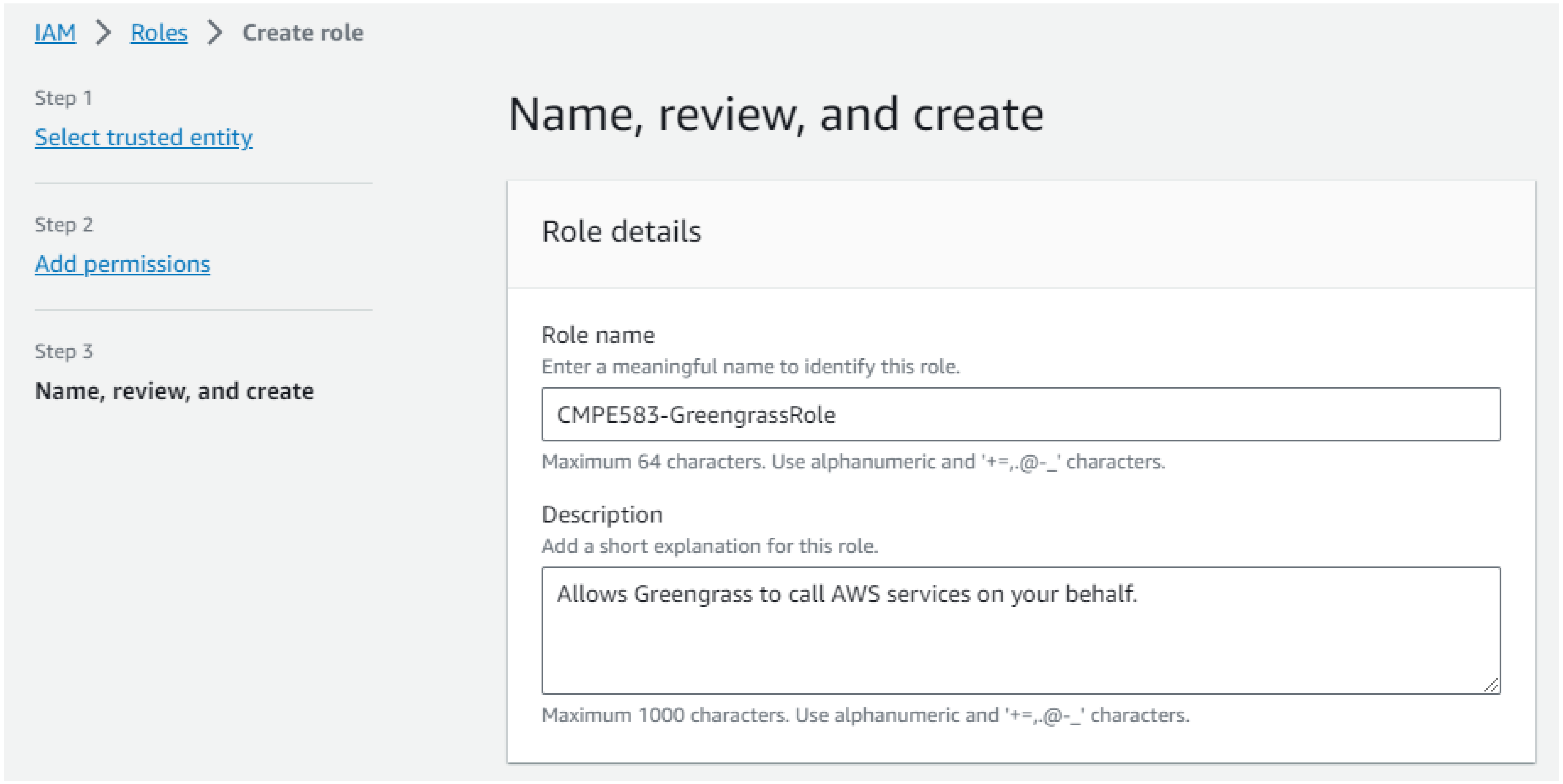
green | All types | 6 matches

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

Set permissions boundary - *optional*

Troubleshooting

Review and Create Role



Name, review, and create

Role details

Role name

Enter a meaningful name to identify this role.

CMPE583-GreengrassRole

Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description

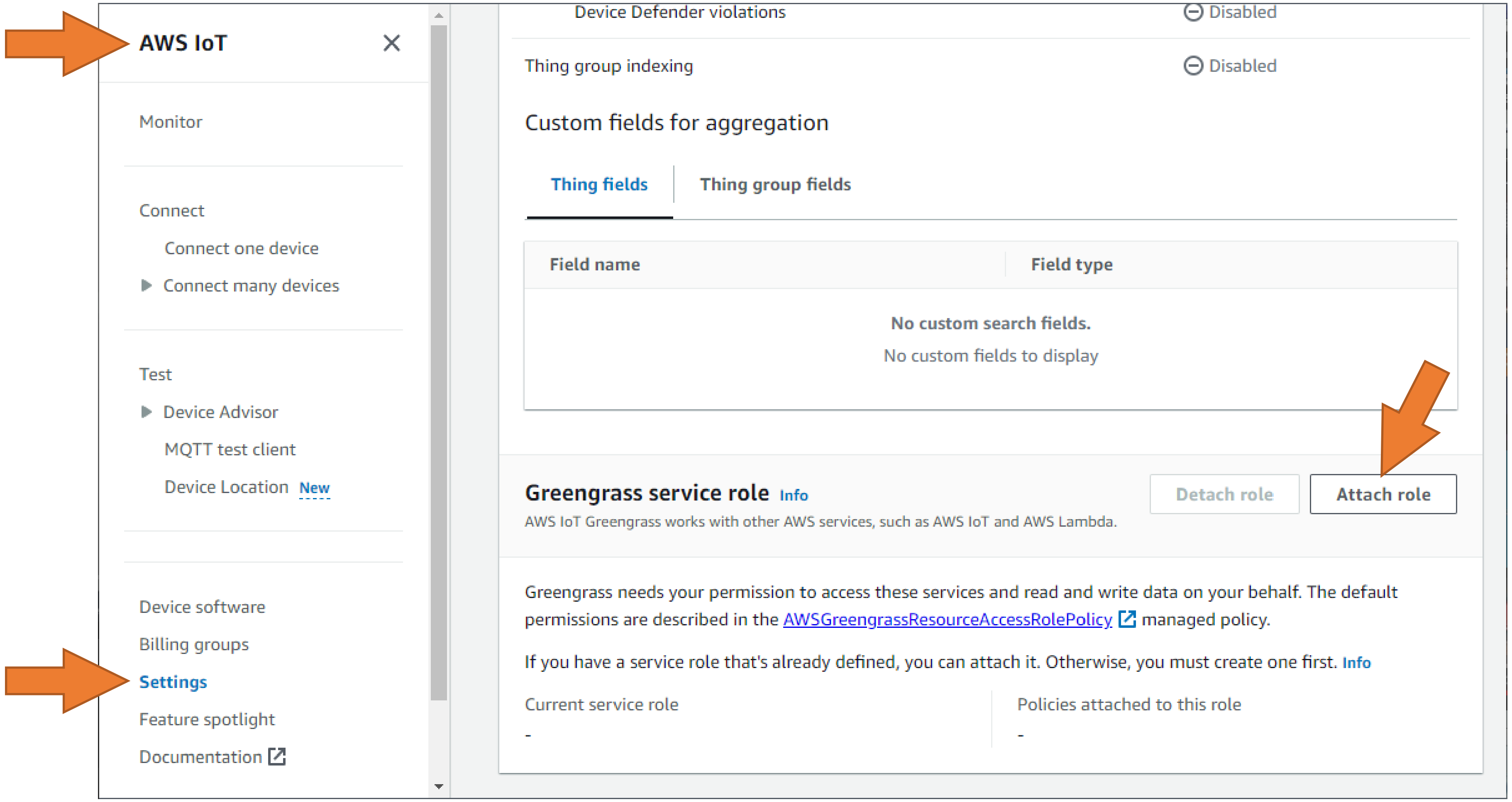
Add a short explanation for this role.

Allows Greengrass to call AWS services on your behalf.

Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.

Troubleshooting

Attach Role to AWS IoT Service



Device Defender violations ⊖ Disabled

Thing group indexing ⊖ Disabled

Custom fields for aggregation

Field name	Field type
No custom search fields. No custom fields to display	

Greengrass service role Info

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

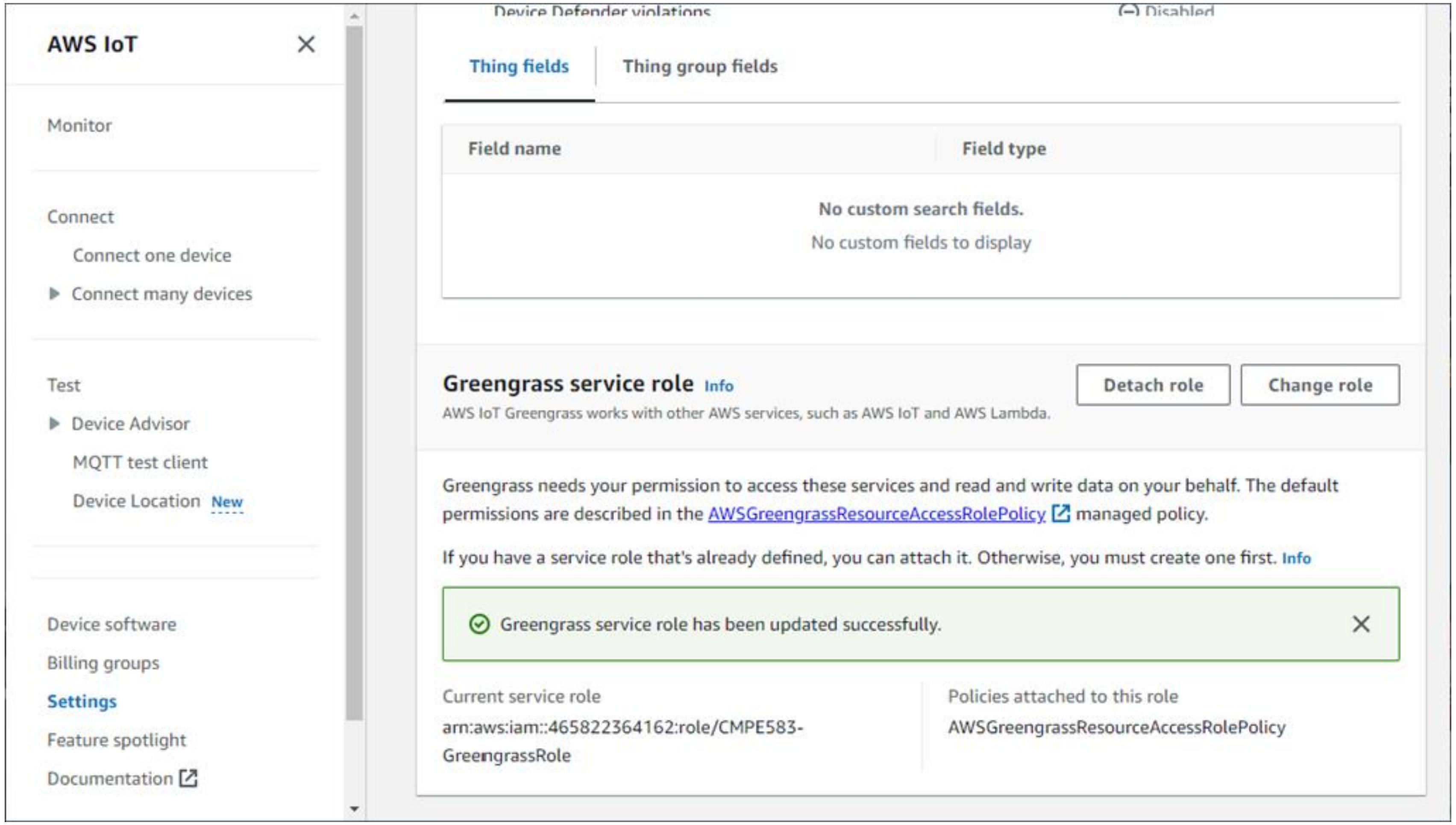
Greengrass needs your permission to access these services and read and write data on your behalf. The default permissions are described in the AWSGreengrassResourceAccessRolePolicy managed policy.

If you have a service role that's already defined, you can attach it. Otherwise, you must create one first. Info

Current service role	Policies attached to this role
-	-

Troubleshooting

Attach Role to AWS IoT Service



Field name	Field type
No custom search fields. No custom fields to display	

Greengrass service role Info

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

Greengrass needs your permission to access these services and read and write data on your behalf. The default permissions are described in the AWSGreengrassResourceAccessRolePolicy managed policy.

If you have a service role that's already defined, you can attach it. Otherwise, you must create one first. Info

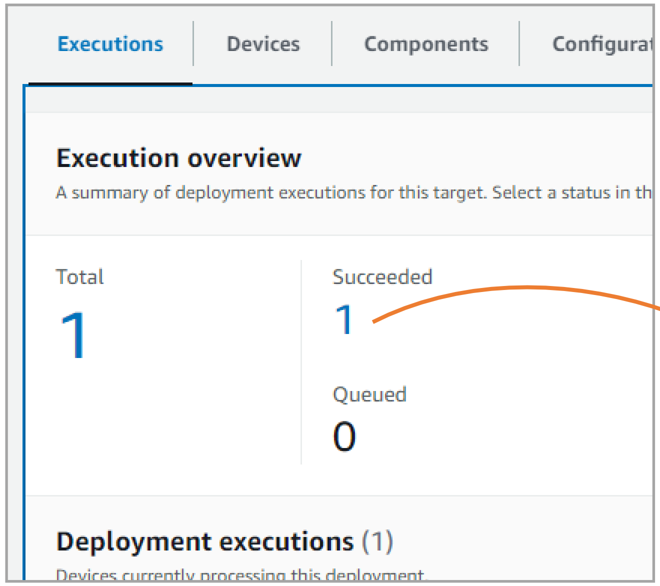
Greengrass service role has been updated successfully.

Current service role	Policies attached to this role
arn:aws:iam::465822364162:role/CMPE583-GreengrassRole	AWSGreengrassResourceAccessRolePolicy

Troubleshooting

Verify the Deployment

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



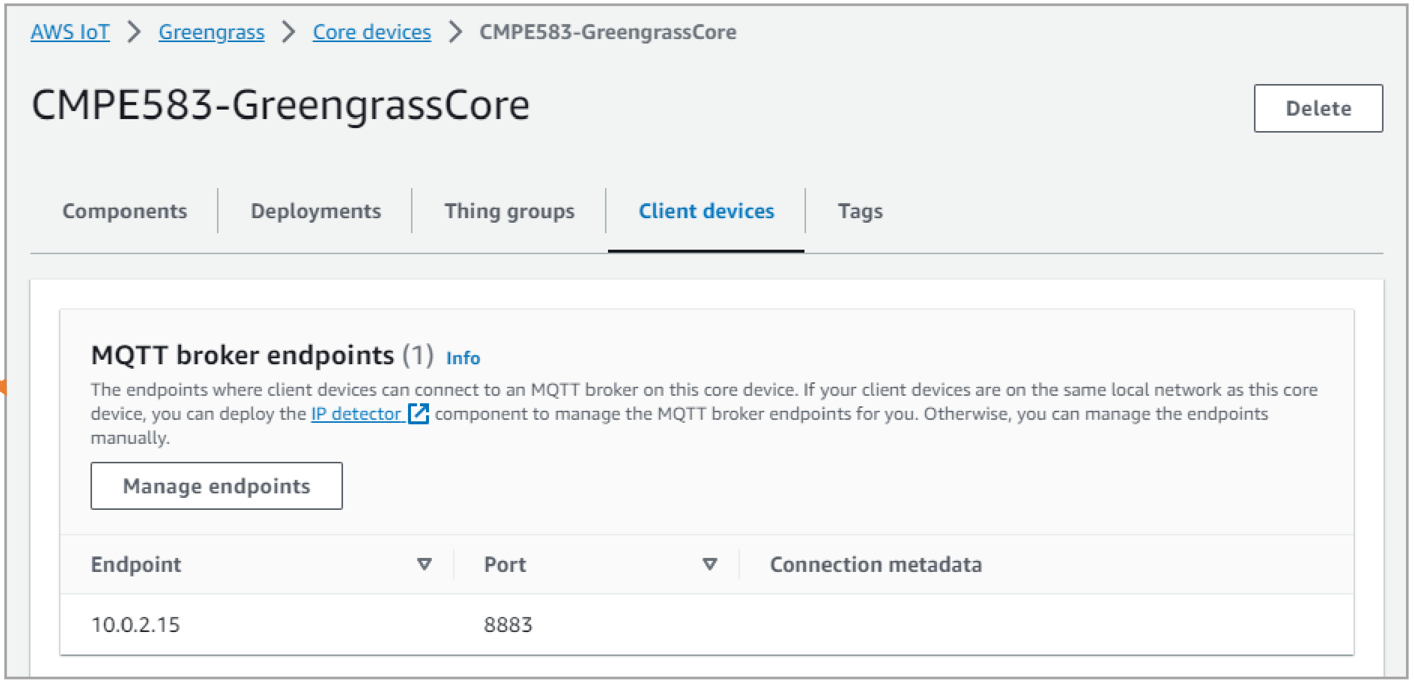
Execution overview

A summary of deployment executions for this target. Select a status in th

Total	Succeeded
1	1
	Queued
	0

Deployment executions (1)

Devices currently processing this deployment



CMPE583-GreengrassCore

MQTT broker endpoints (1)

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

Endpoint	Port	Connection metadata
10.0.2.15	8883	

Publish/Subscribe AWS IoT Core MQTT Messages

AWS IoT Greengrass Core IPC

- Components running on your core device can use the AWS IoT Greengrass Core interprocess communication (IPC) library in the AWS IoT Device SDK to communicate with the AWS IoT Greengrass nucleus and other Greengrass components.
- 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.

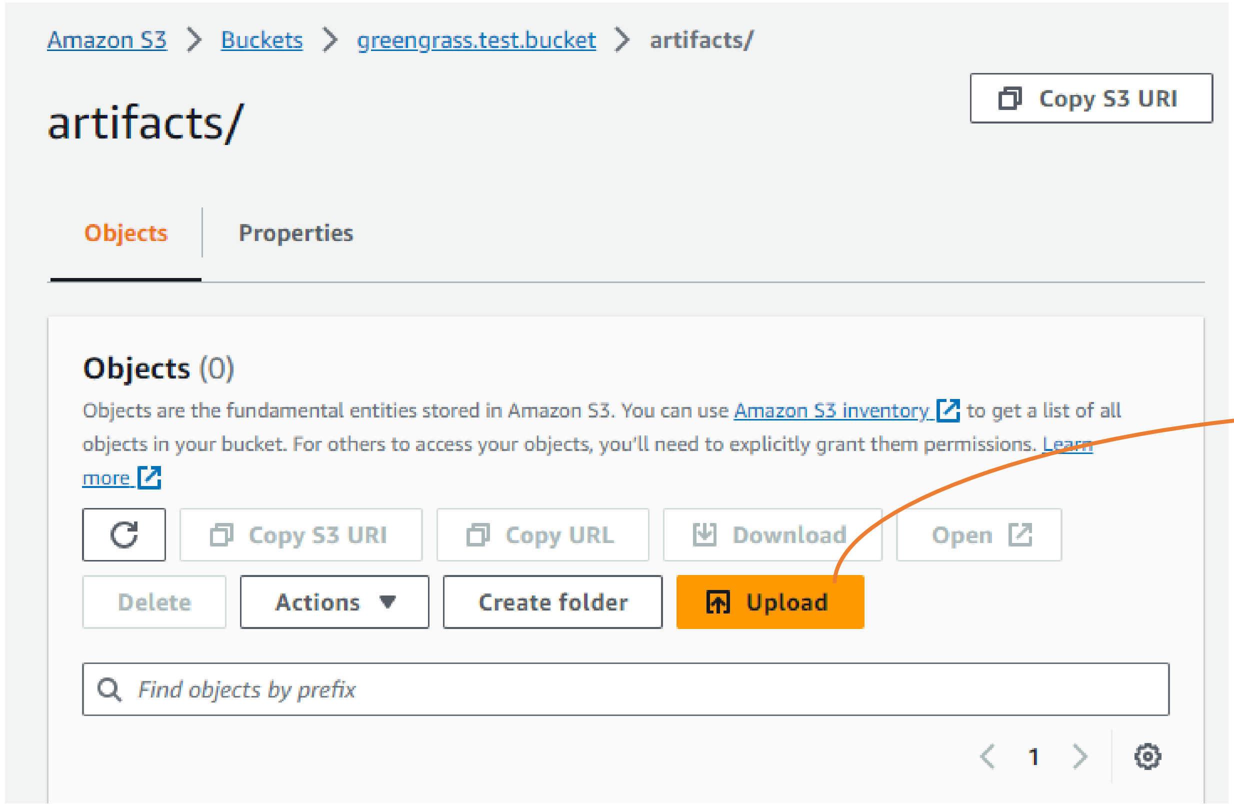
MQTT

- 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

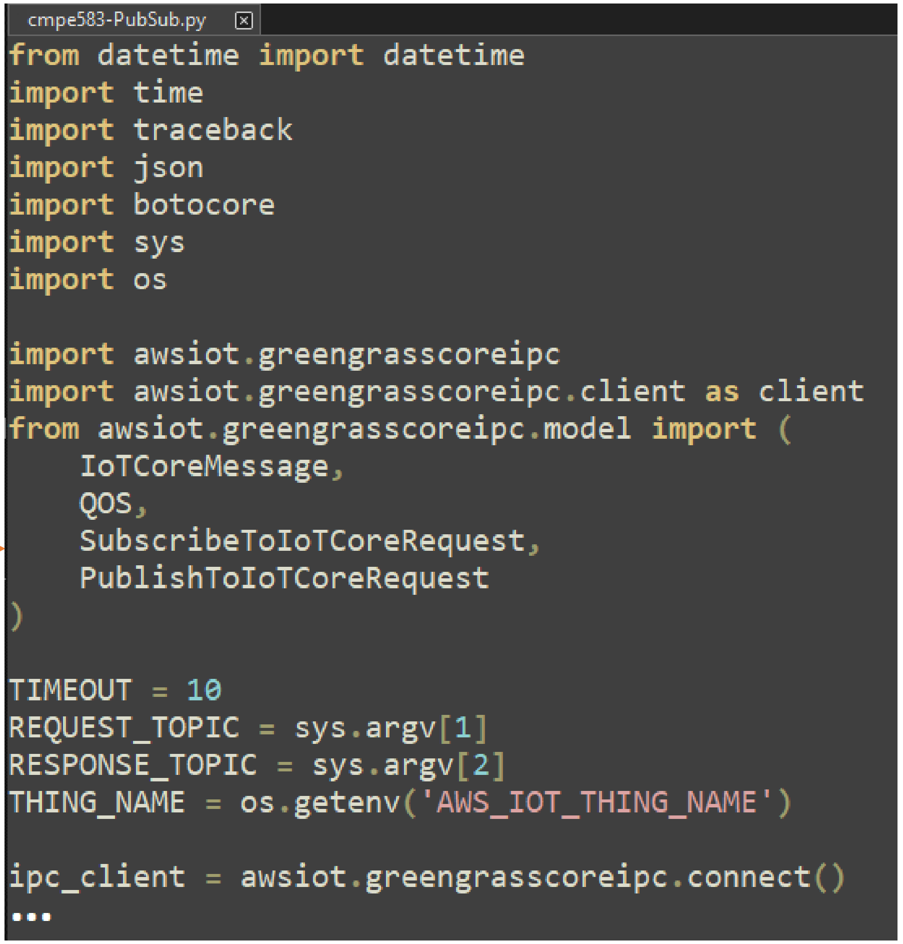


Amazon S3 > Buckets > greengrass.test.bucket > artifacts/

artifacts/

Objects (0)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

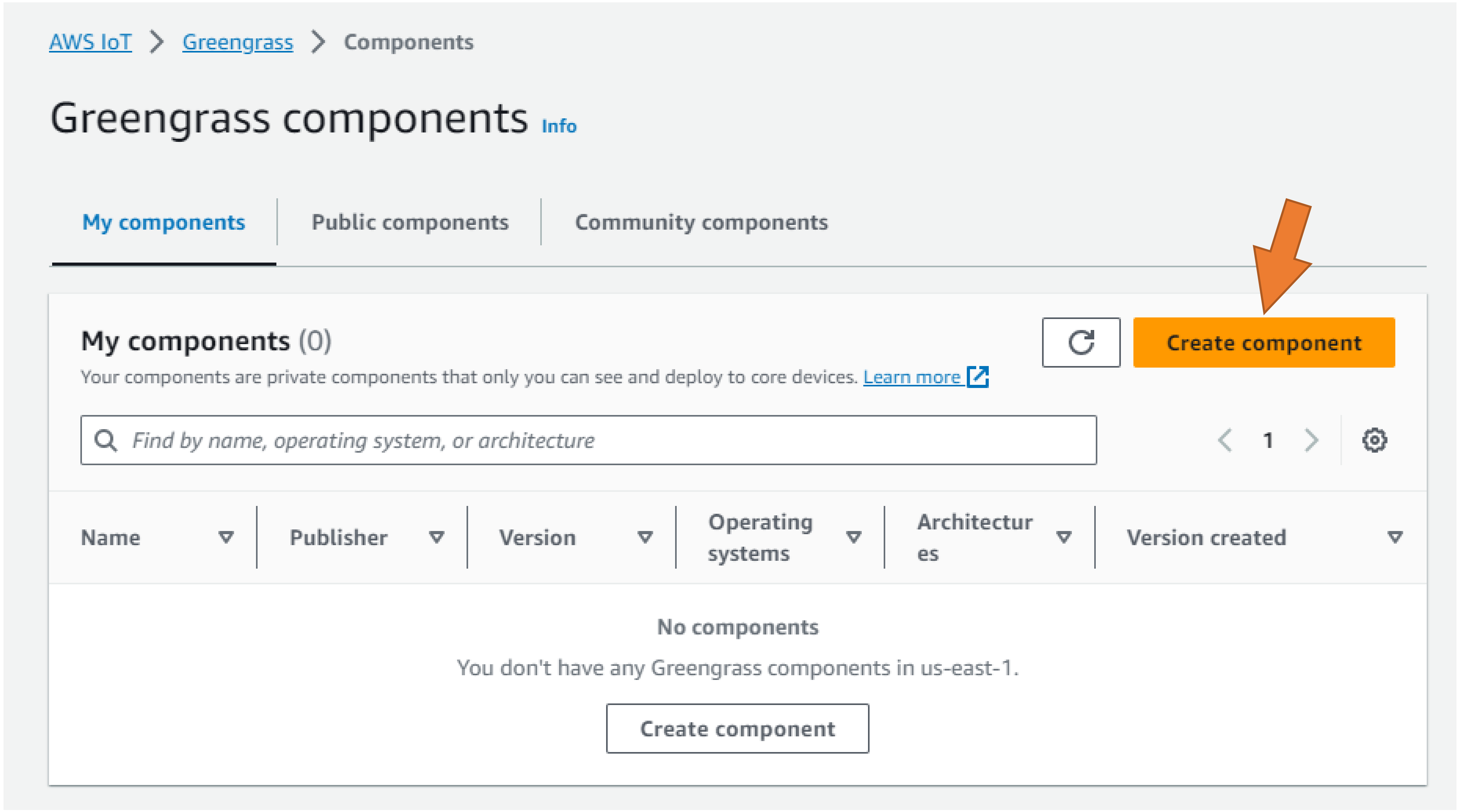


cmpe583-PubSub.py

```python
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)



AWS IoT > Greengrass > Components

Greengrass components Info

My components | Public components | Community components

My components (0)

Your components are private components that only you can see and deploy to core devices. Learn more

Create component

Find by name, operating system, or architecture

Name	Publisher	Version	Operating systems	Architectures	Version created

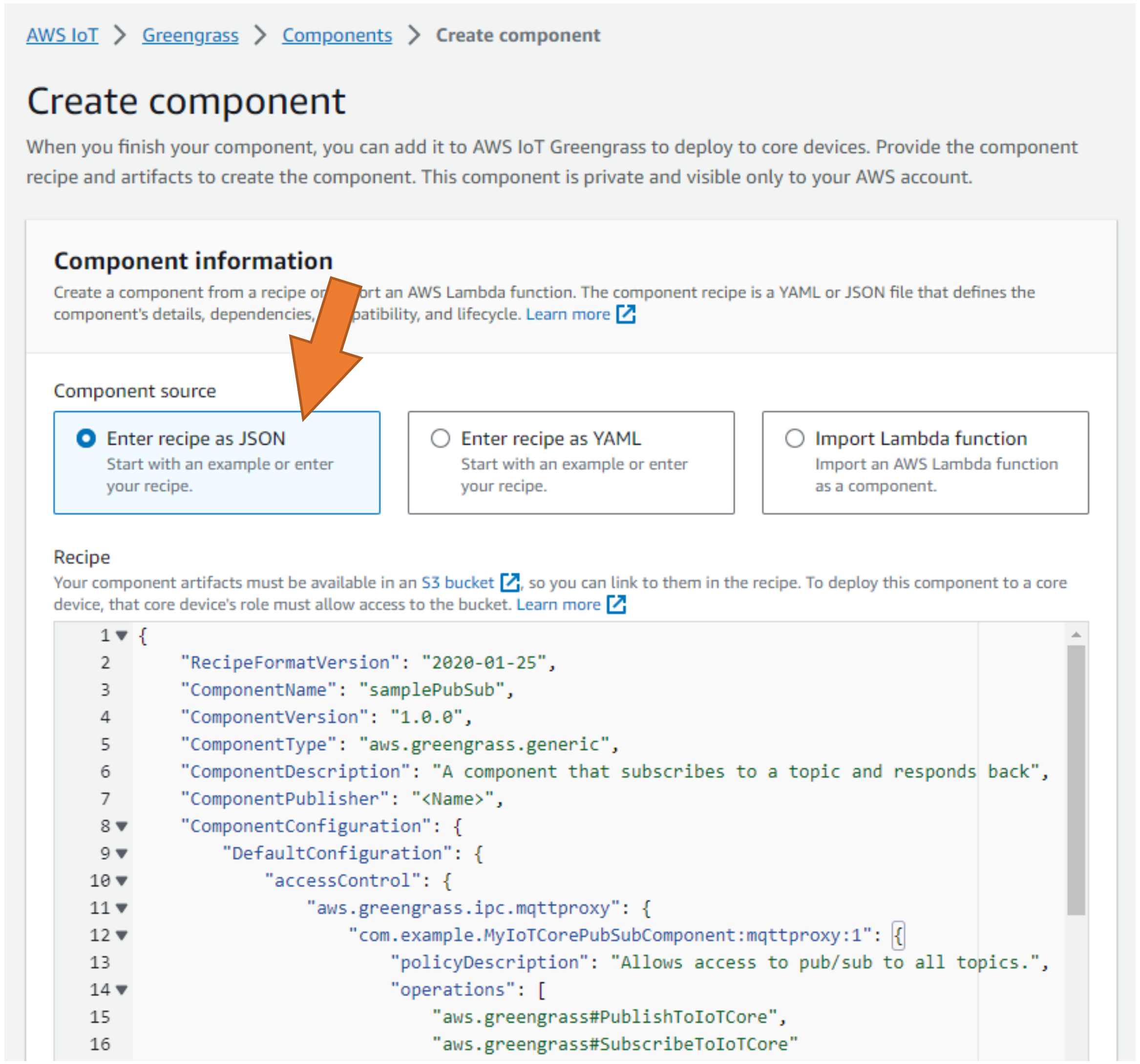
No components

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

Create component

Step 2: Create a Custom Component

Enter the Recipe



AWS IoT > Greengrass > Components > **Create component**

Create component

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

Component information

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

Component source

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

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

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

Recipe

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

```
{
    "RecipeFormatVersion": "2020-01-25",
    "ComponentName": "samplePubSub",
    "ComponentVersion": "1.0.0",
    "ComponentType": "aws.greengrass.generic",
    "ComponentDescription": "A component that subscribes to a topic and responds back",
    "ComponentPublisher": "<Name>",
    "ComponentConfiguration": {
        "DefaultConfiguration": {
            "accessControl": {
                "aws.greengrass.ipc.mqttproxy": {
                    "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
                        "policyDescription": "Allows access to pub/sub to all topics.",
                        "operations": [
                            "aws.greengrass#PublishToIoTCore",
                            "aws.greengrass#SubscribeToIoTCore"
```

Step 2: Create a Custom Component

Adjust the Access Control

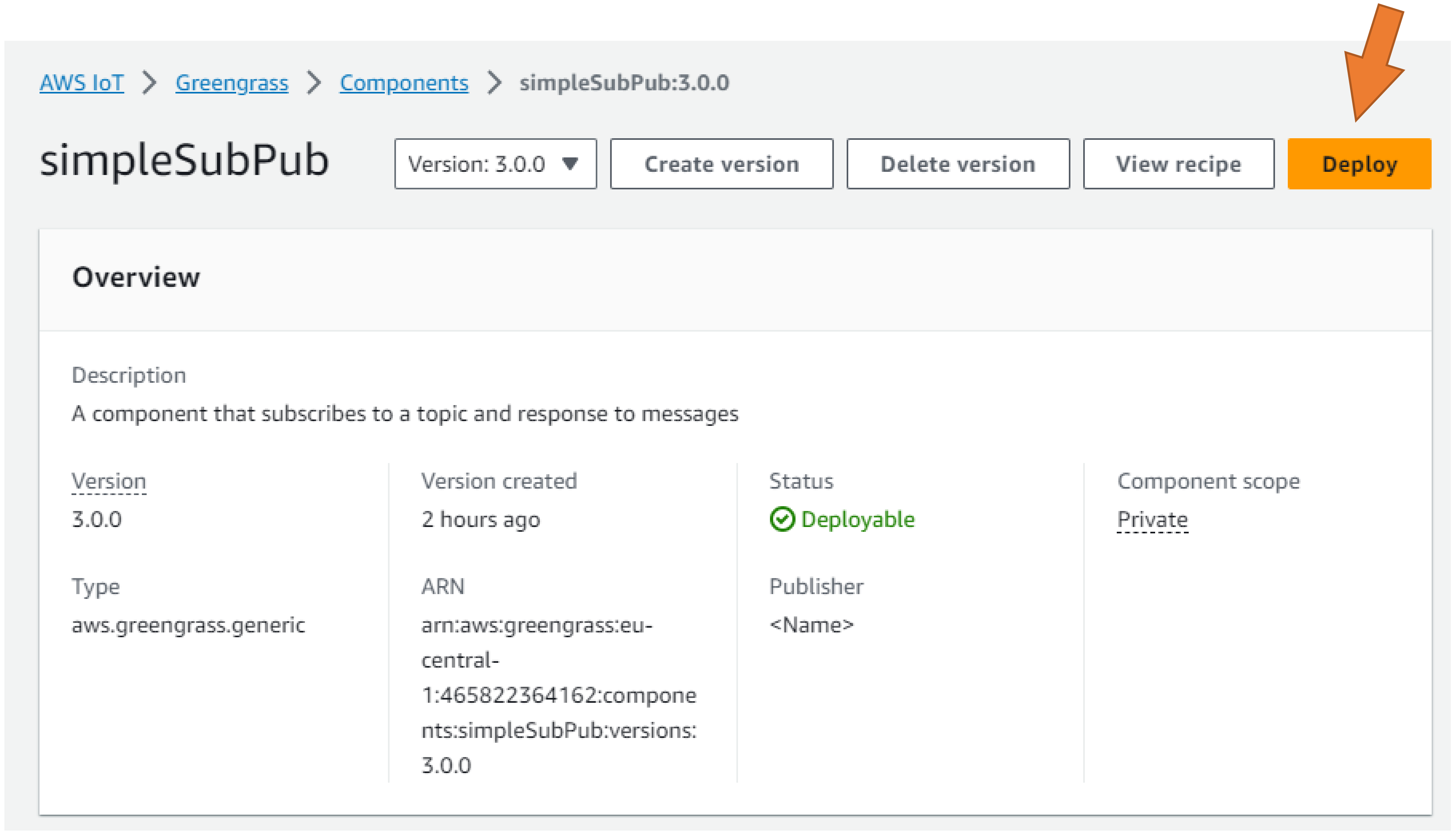
```
{
    "accessControl": {
       "aws.greengrass.ipc.mqttproxy": {
         "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
            "policyDescription": "Allows access to publish/subscribe to all topics.",
            "operations": [
               "aws.greengrass#PublishToIoTCore",
               "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)



simpleSubPub

Overview

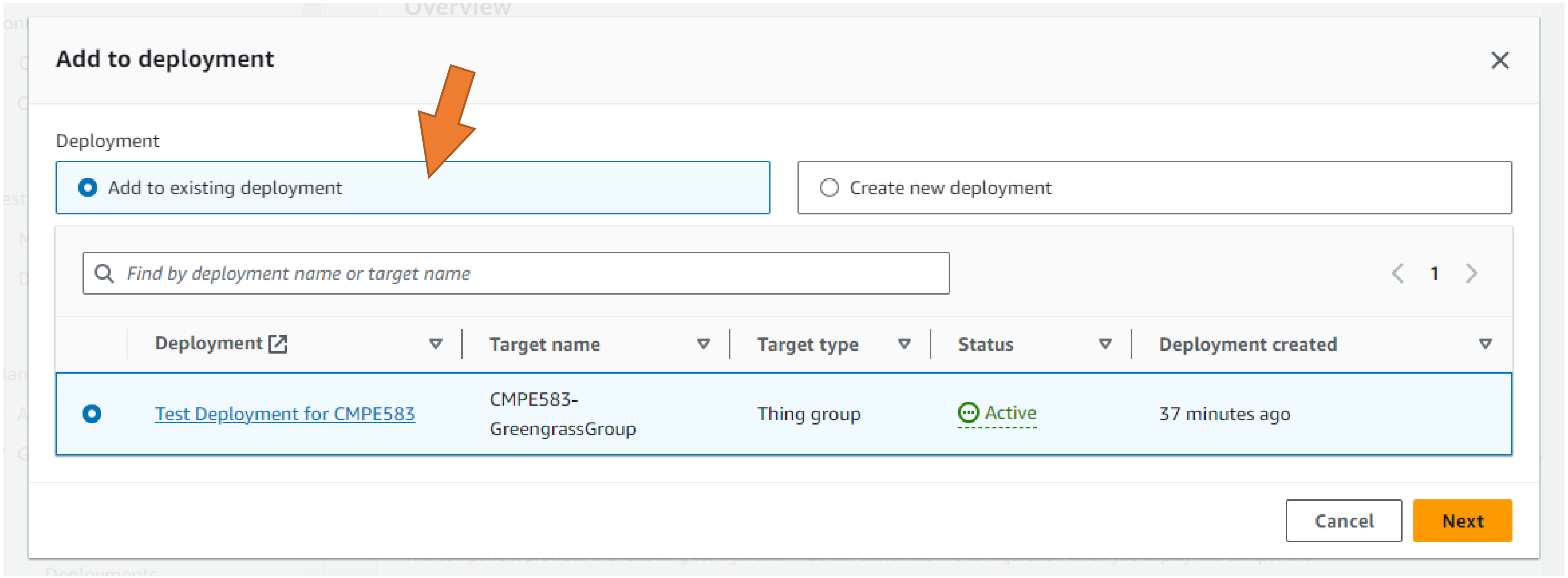
Description
A component that subscribes to a topic and response to messages

Version	Version created	Status	Component scope
3.0.0	2 hours ago	Deployable	Private

Type	ARN	Publisher
aws.greengrass.generic	arn:aws:greengrass:eu-central-1:465822364162:components:simpleSubPub:versions:3.0.0	<Name>

Step 3: Deploy Your Component

Add Custom Component to Existing Deployment



Add to deployment

Deployment

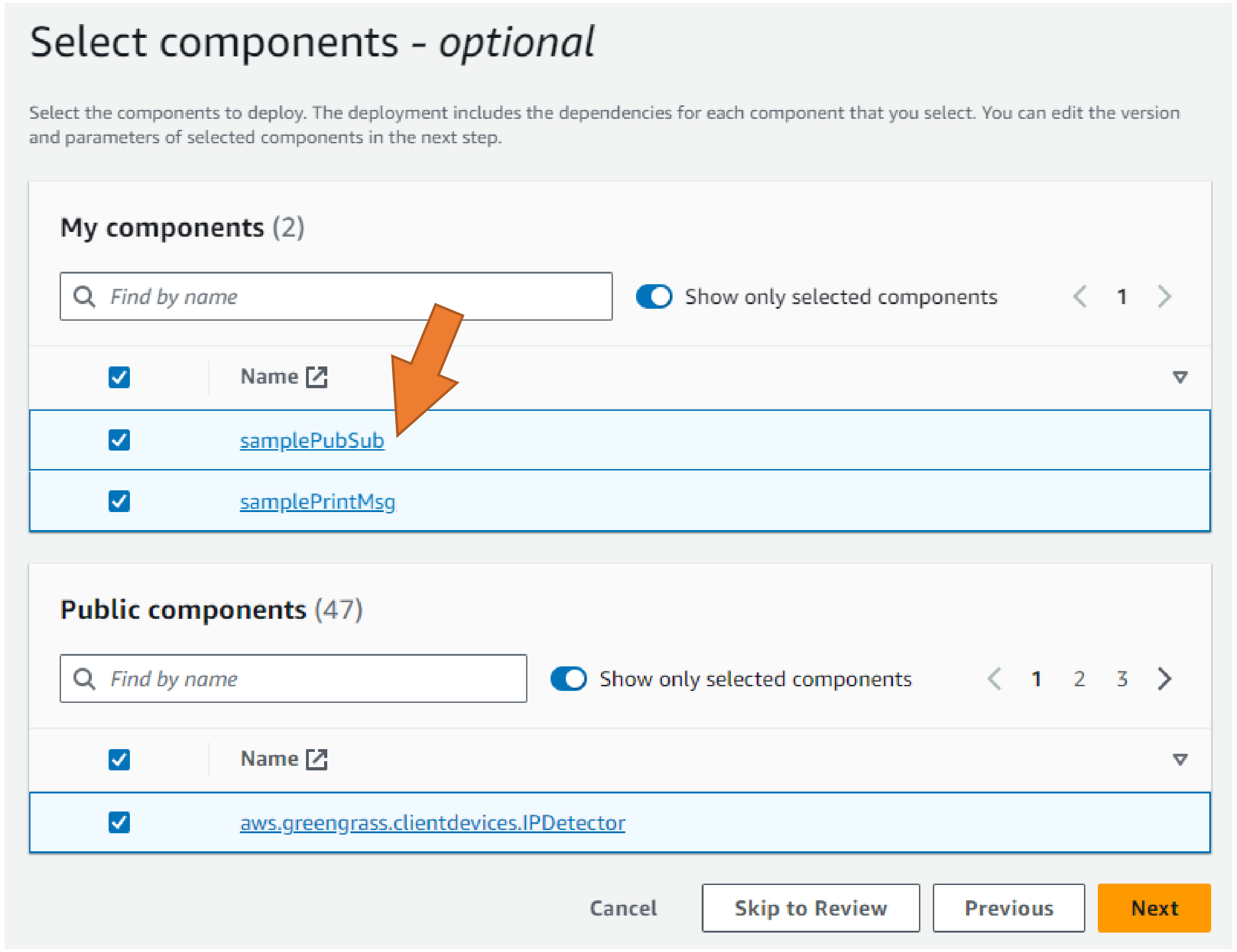
[✓ Add to existing deployment] Create new deployment

Find by deployment name or target name

	Deployment	Target name	Target type	Status	Deployment created
●	Test Deployment for CMPE583	CMPE583-GreengrassGroup	Thing group	Active	37 minutes ago

Step 3: Deploy Your Component

Select the Component



Select components - *optional*

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

My components (2)

Find by name

Show only selected components: on

✓	Name
✓	samplePubSub
✓	samplePrintMsg

Public components (47)

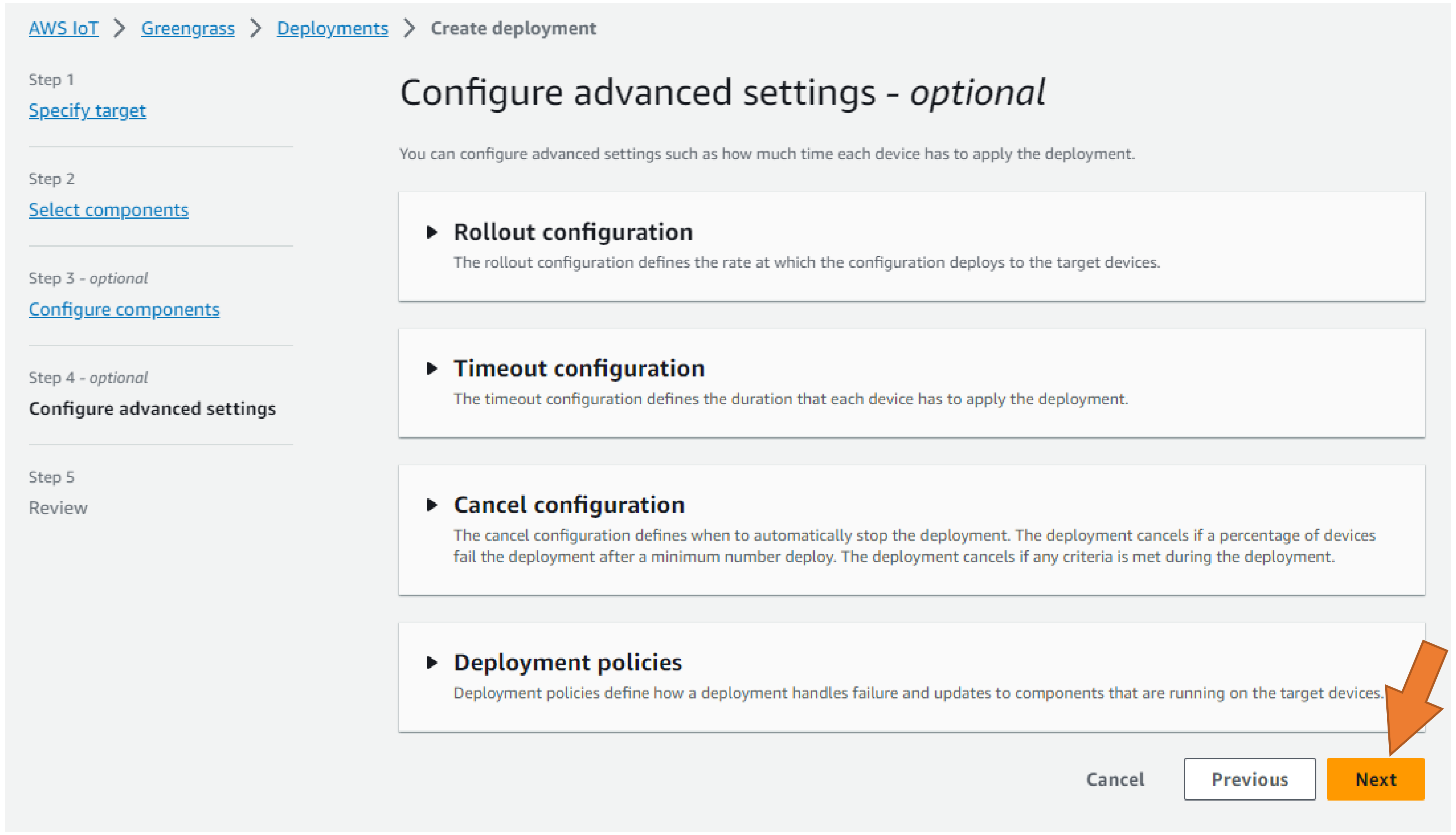
Find by name

Show only selected components: on

✓	Name
✓	aws.greengrass.clientdevices.IPDetector

Step 3: Deploy Your Component

Configure Advanced Settings



AWS IoT > Greengrass > Deployments > Create deployment

Step 1
Specify target

Step 2
Select components

Step 3 - *optional*
Configure components

Step 4 - *optional*
Configure advanced settings

Step 5
Review

Configure advanced settings - *optional*

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

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

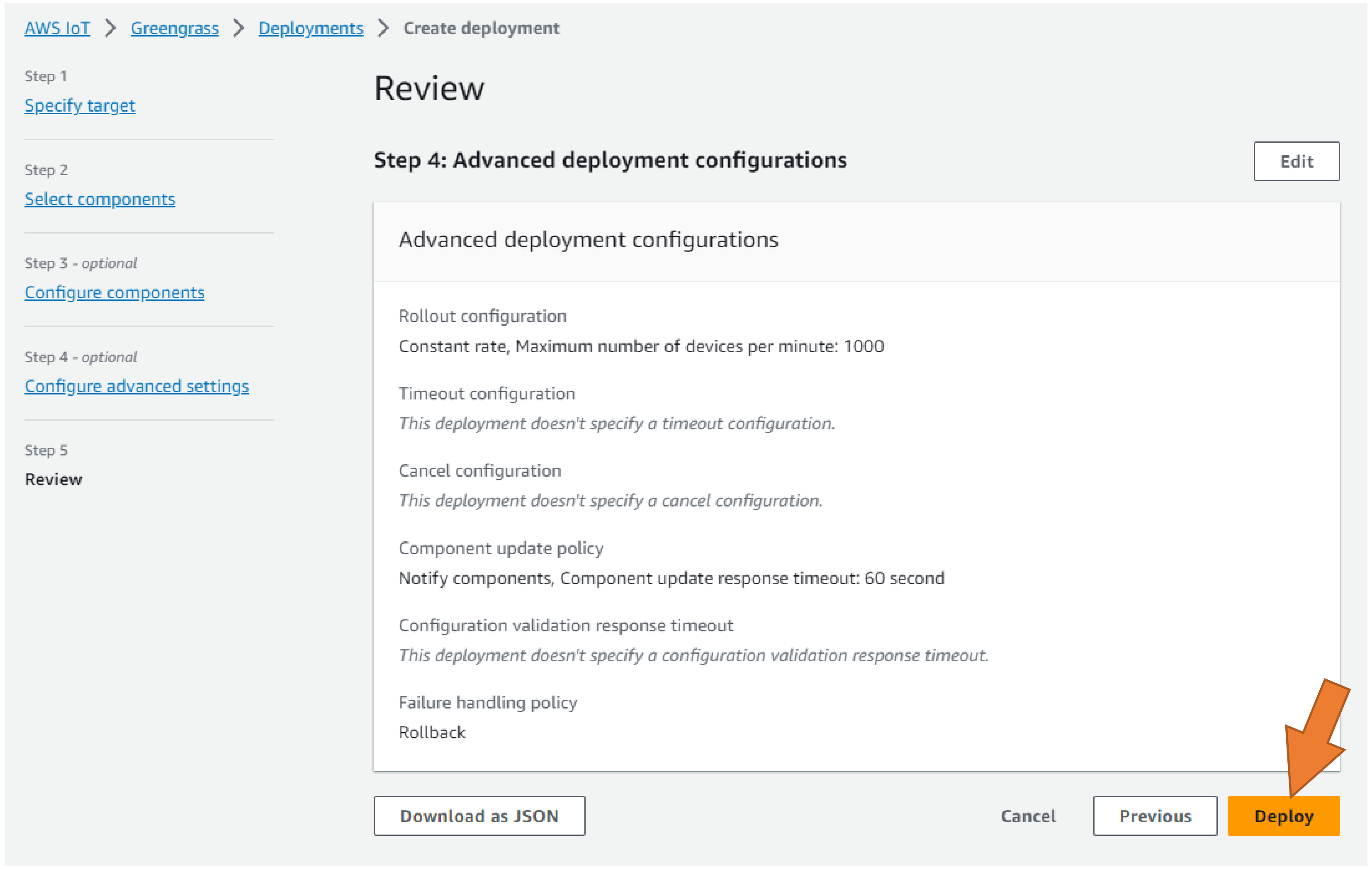
▶ **Timeout configuration**
The timeout configuration defines the duration that each device has to apply the deployment.

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

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

Step 3: Deploy Your Component

Review & Deploy



AWS IoT > Greengrass > Deployments > Create deployment

Step 1
Specify target

Step 2
Select components

Step 3 - *optional*
Configure components

Step 4 - *optional*
Configure advanced settings

Step 5
Review

Review

Step 4: Advanced deployment configurations

Edit

Advanced deployment configurations

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

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

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

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

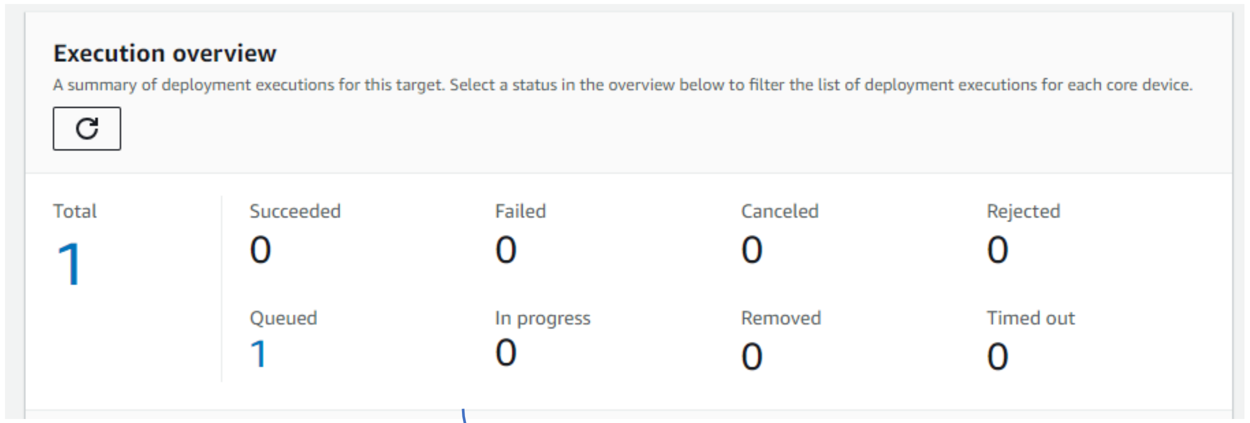
Configuration validation response timeout
This deployment doesn't specify a configuration validation response timeout.

Failure handling policy
Rollback

Download as JSON | Cancel | Previous | Deploy

Step 4: Verify the Component

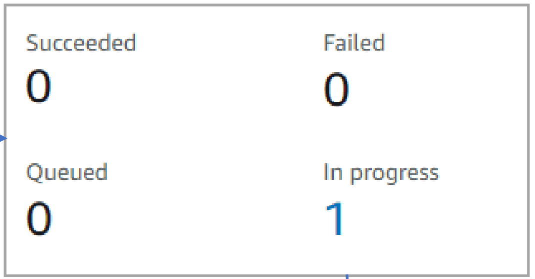
Check AWS IoT Greengrass Console



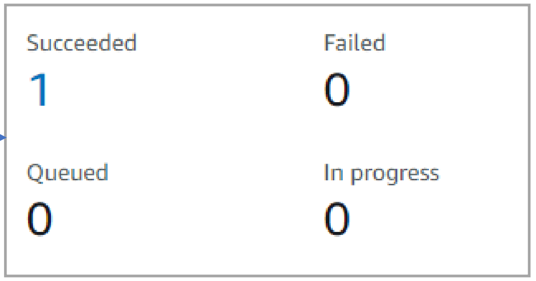
Execution overview

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

Total	Succeeded	Failed	Canceled	Rejected
1	0	0	0	0
	Queued	In progress	Removed	Timed out
	1	0	0	0



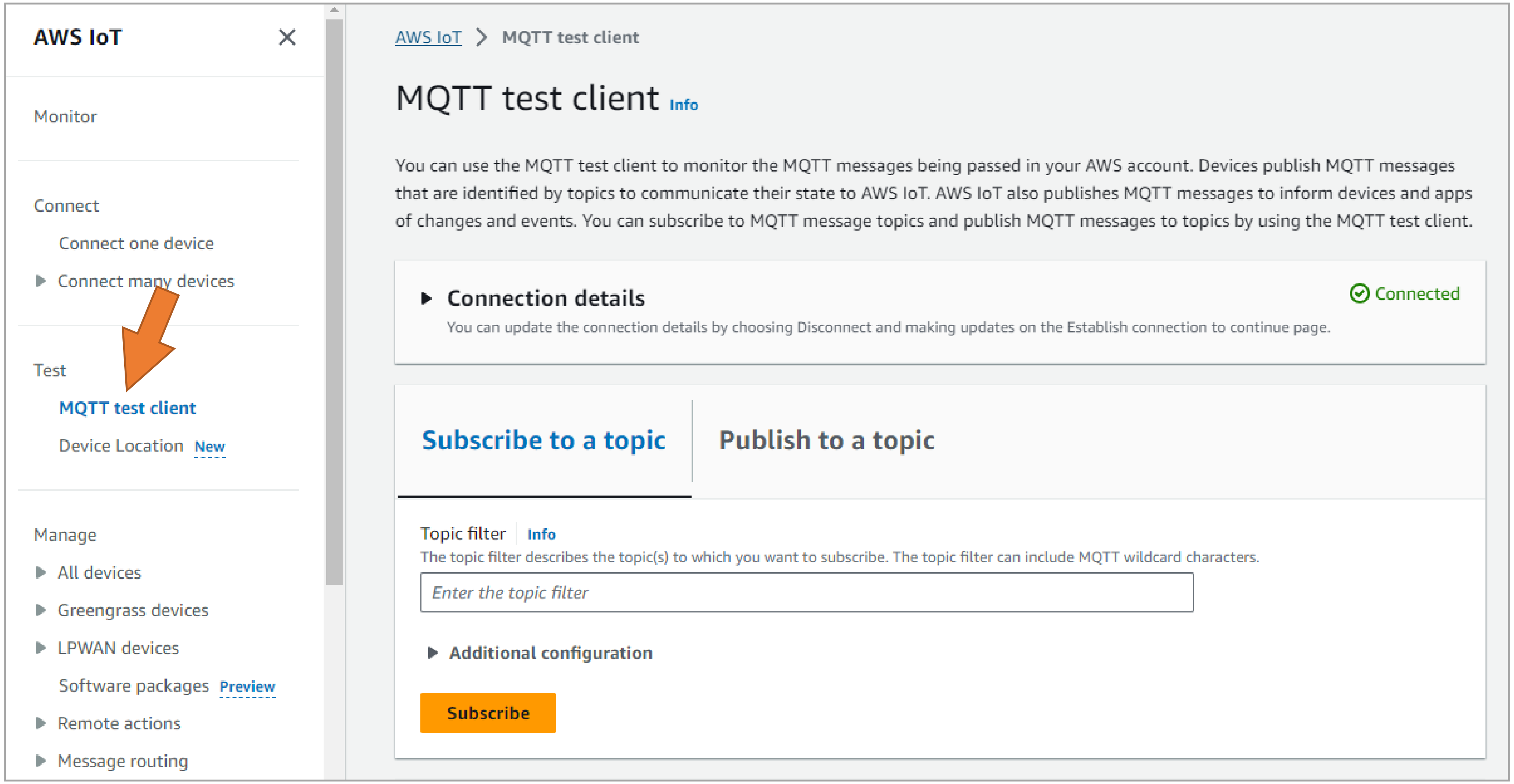
Succeeded	Failed
0	0
Queued	In progress
0	1



Succeeded	Failed
1	0
Queued	In progress
0	0

Test Your Deployment

Via AWS MQTT Test Client



MQTT test client Info

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

Connection details — Connected

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

Subscribe to a topic | Publish to a topic

Topic filter Info

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

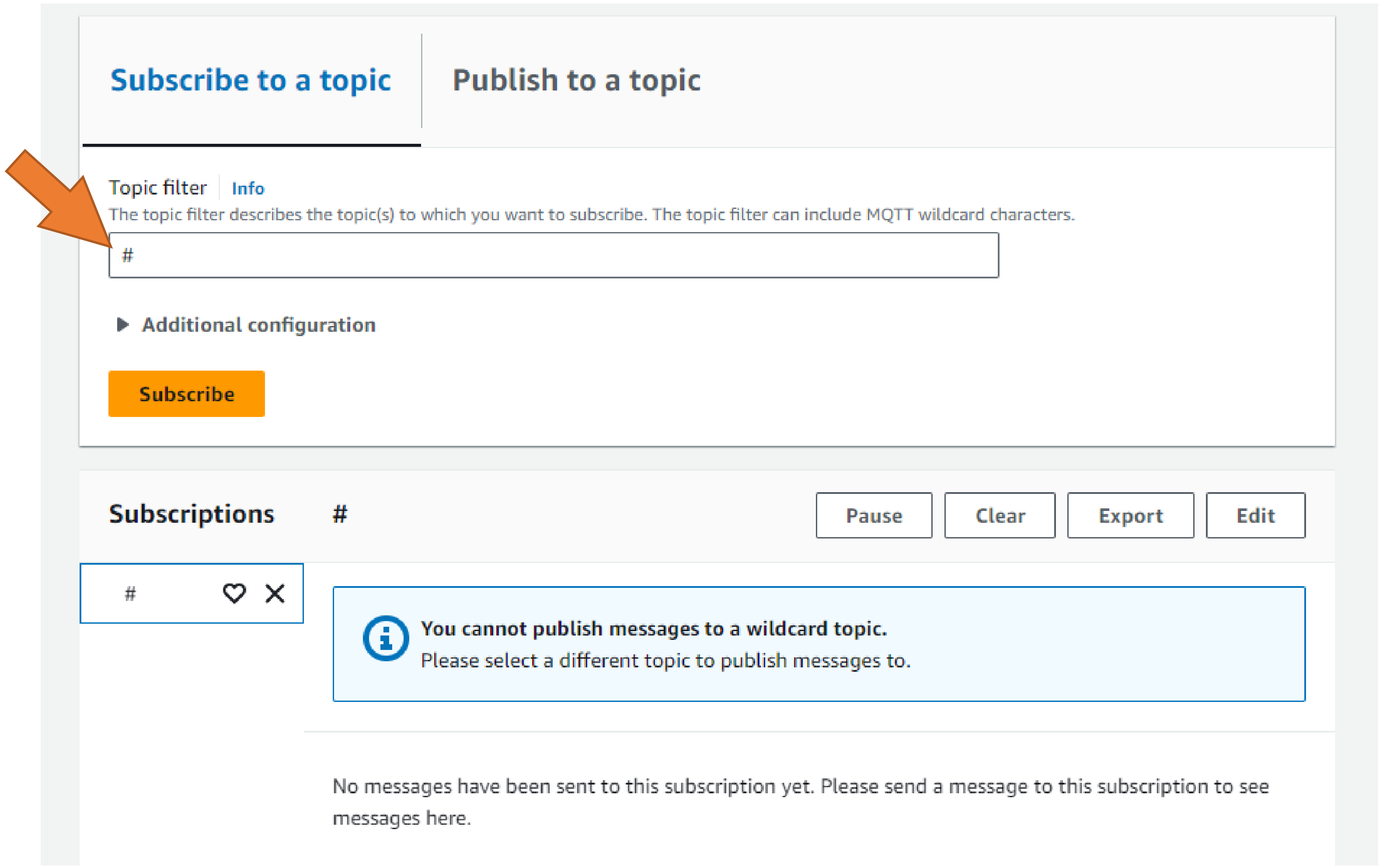
Enter the topic filter

Additional configuration

Subscribe

Test Your Deployment

Subscribe All Topics



Topic filter

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

#

Subscriptions #

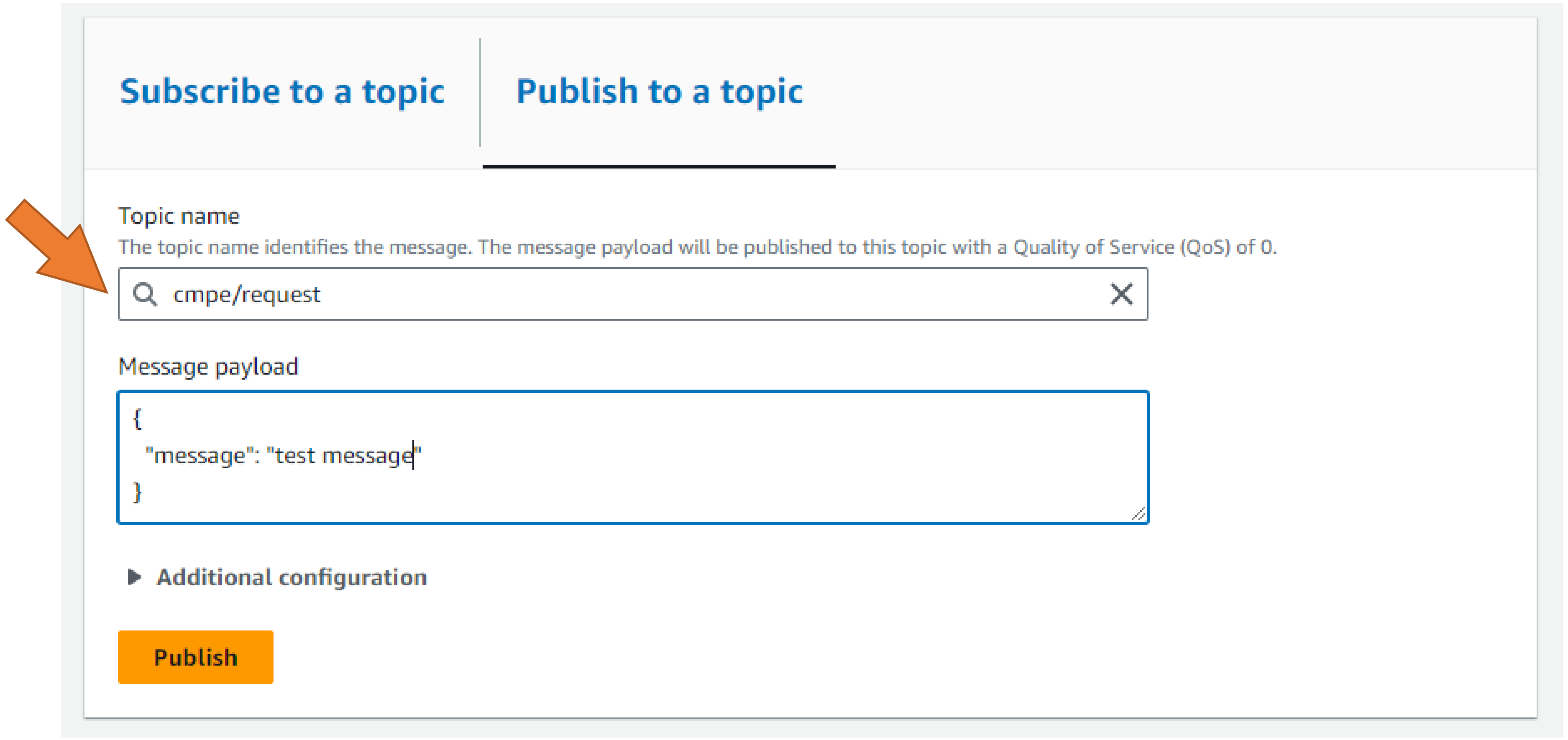
#

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

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

Test Your Deployment

Send a Test Message



Topic name

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

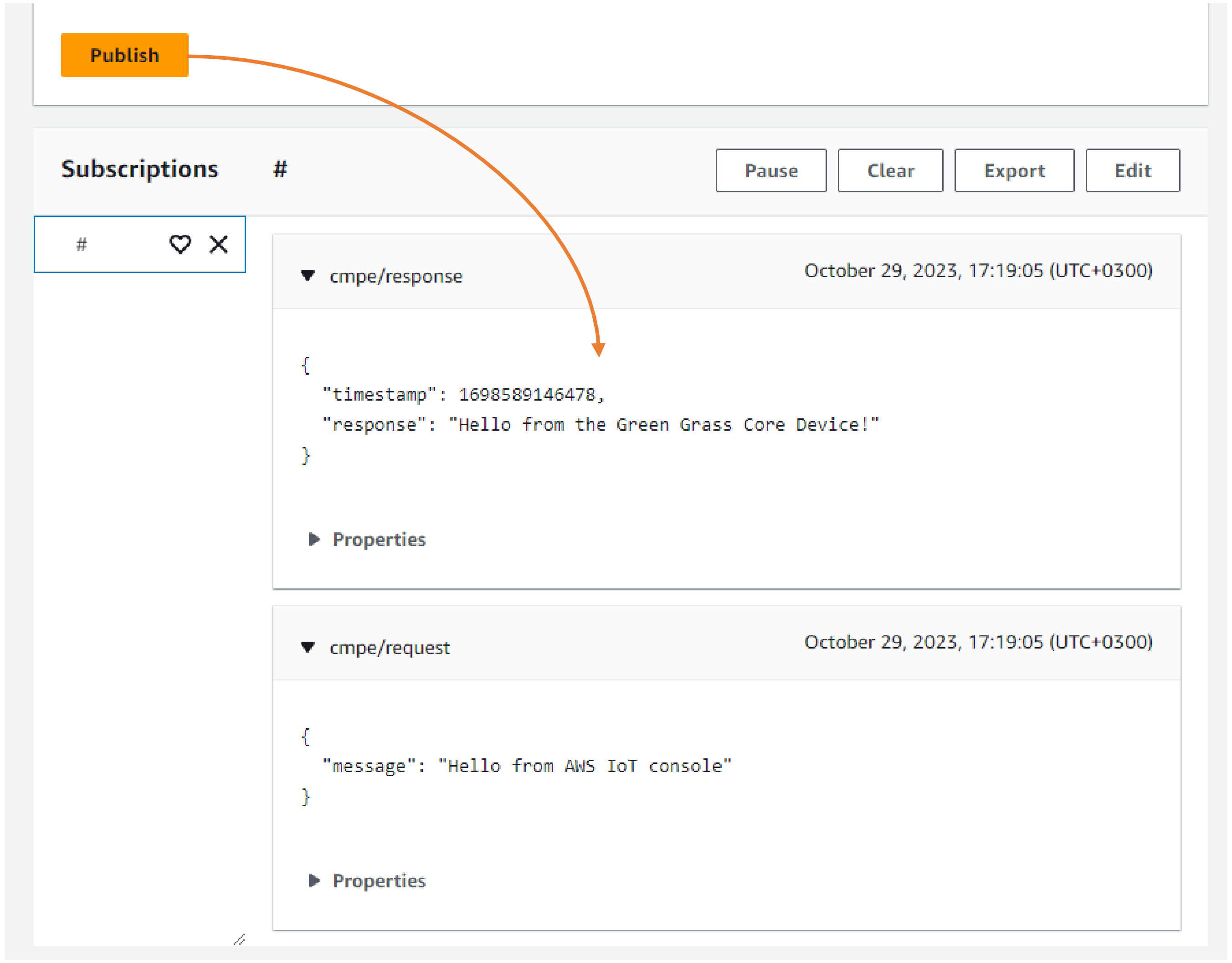
cmpe/request

Message payload

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

Test Your Deployment

Check If the Client Responds Back



cmpe/response October 29, 2023, 17:19:05 (UTC+0300)

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

cmpe/request October 29, 2023, 17:19:05 (UTC+0300)

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

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

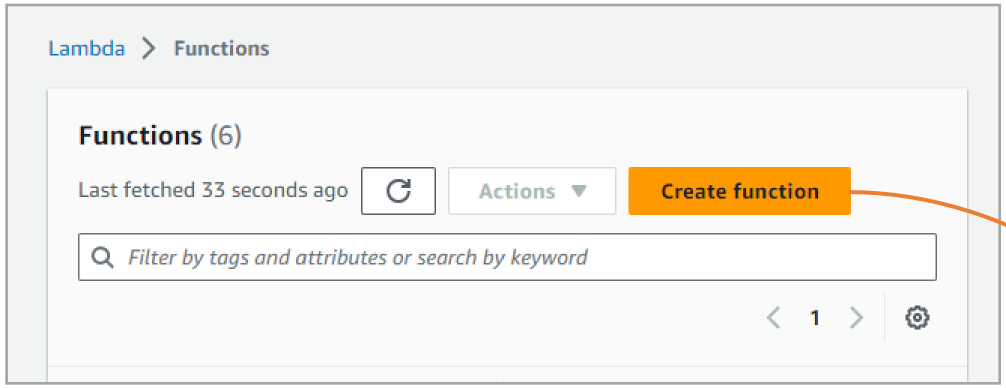
- On-demand functions start when they are invoked and stop when there are no tasks left to run.
- Each invocation of the function creates a separate container, also called a sandbox, to process invocations, unless an existing container is available for reuse.
- Any of the containers might process data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

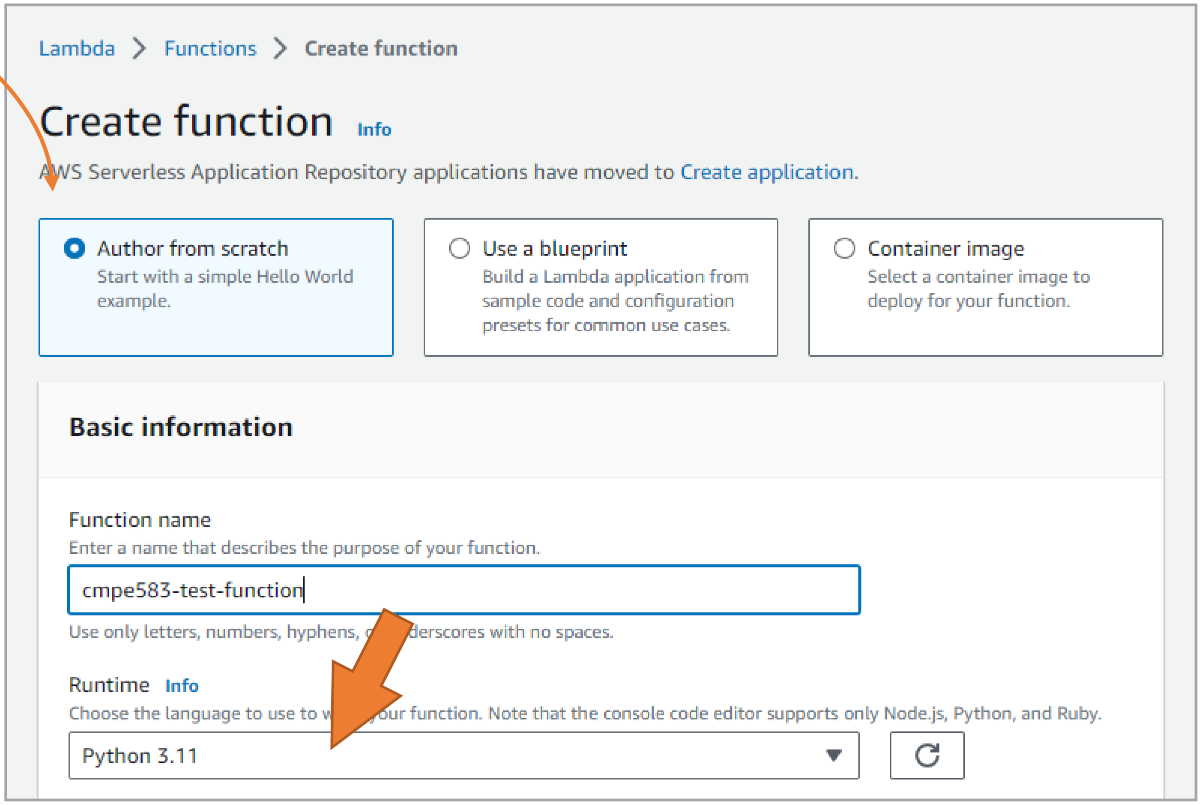
Via AWS Lambda (Console)



Lambda > Functions

Functions (6)

Last fetched 33 seconds ago



Lambda > Functions > Create function

Create function Info

AWS Serverless Application Repository applications have moved to Create application.

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

Basic information

Function name

Enter a name that describes the purpose of your function.

cmpe583-test-function

Use only letters, numbers, hyphens, or underscores with no spaces.

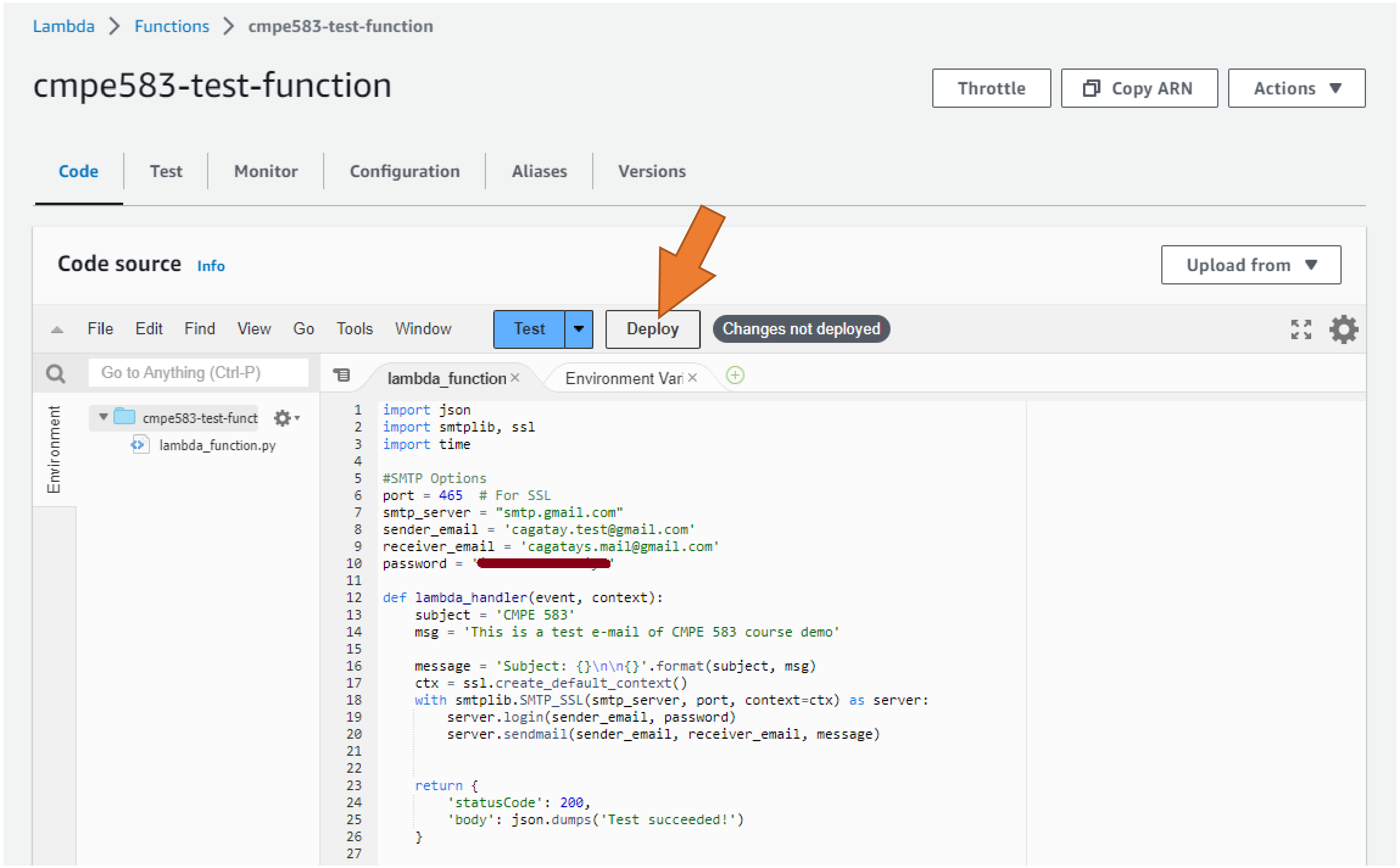
Runtime Info

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

Python 3.11

Step 1: Create AWS Lambda Function

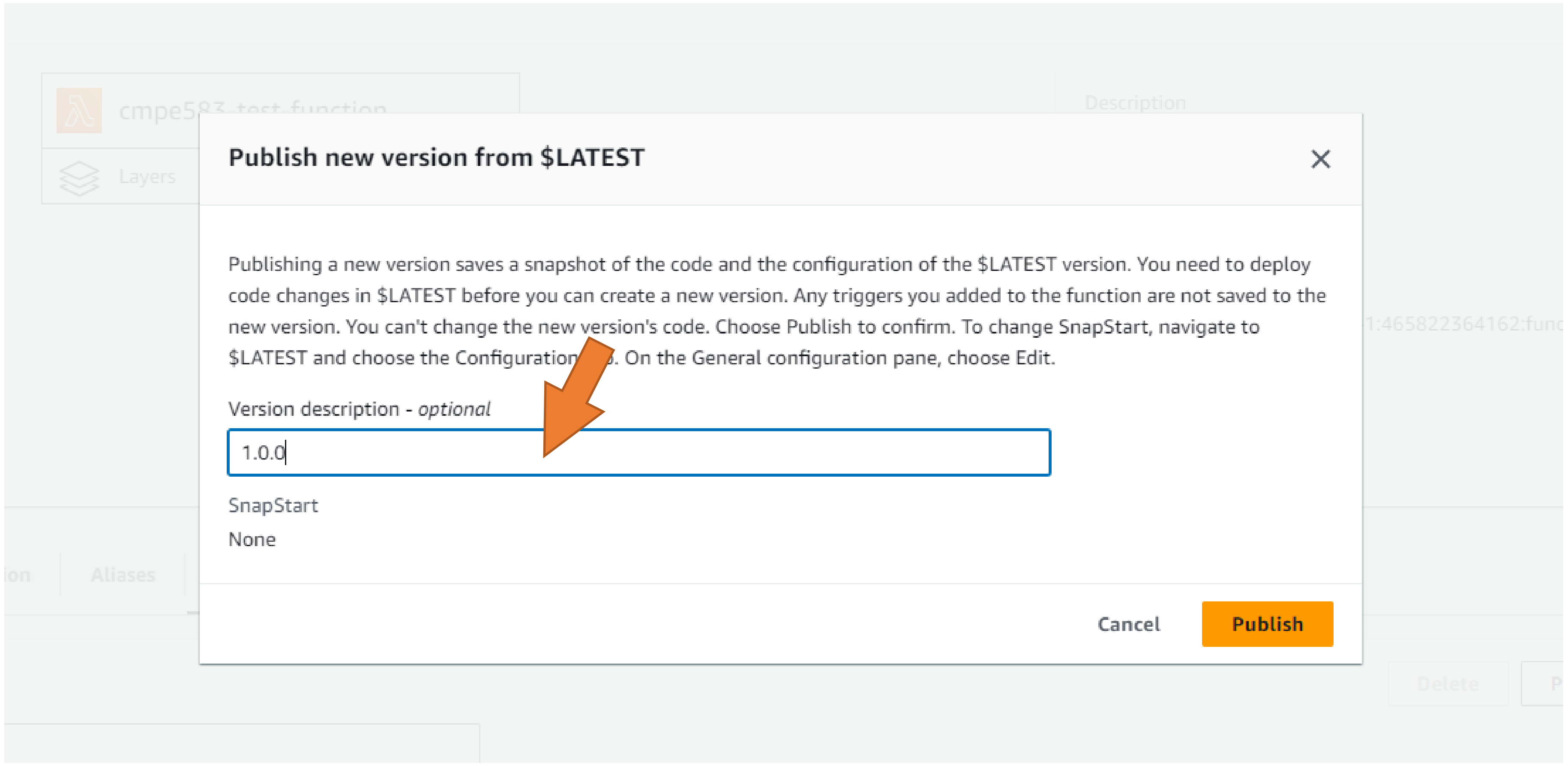
Deploy the Lambda Function



```python
import json
import smtplib, ssl
import time

#SMTP Options
port = 465  # For SSL
smtp_server = "smtp.gmail.com"
sender_email = 'cagatay.test@gmail.com'
receiver_email = 'cagatays.mail@gmail.com'
password = '[redacted]'

def lambda_handler(event, context):
    subject = 'CMPE 583'
    msg = 'This is a test e-mail of CMPE 583 course demo'

    message = 'Subject: {}\n\n{}'.format(subject, msg)
    ctx = ssl.create_default_context()
    with smtplib.SMTP_SSL(smtp_server, port, context=ctx) as server:
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, message)


    return {
        'statusCode': 200,
        'body': json.dumps('Test succeeded!')
    }

```

Step 1: Create AWS Lambda Function

Publish the First Version



Publish new version from $LATEST

Publishing a new version saves a snapshot of the code and the configuration of the $LATEST version. You need to deploy code changes in $LATEST before you can create a new version. Any triggers you added to the function are not saved to the new version. You can't change the new version's code. Choose Publish to confirm. To change SnapStart, navigate to $LATEST and choose the Configuration tab. On the General configuration pane, choose Edit.

Version description - *optional*

1.0.0

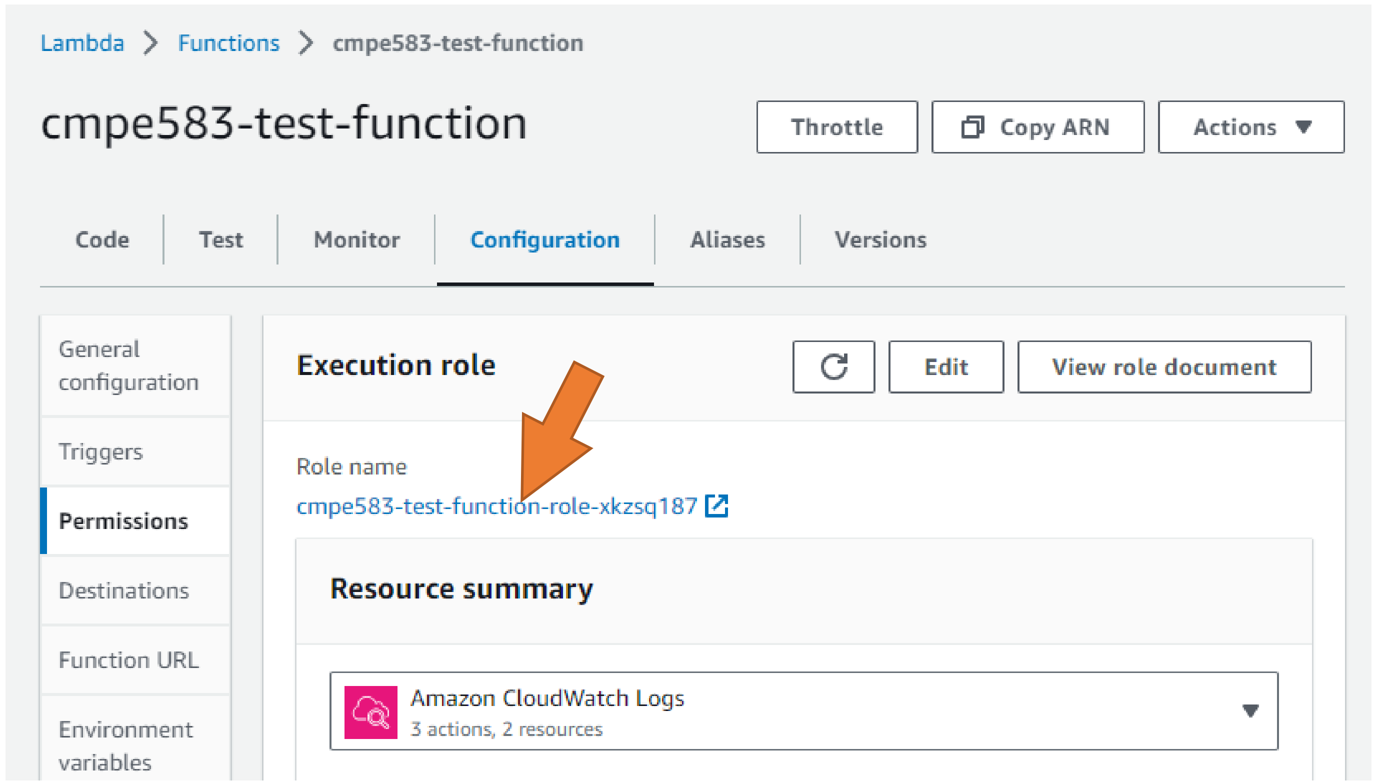
SnapStart

None

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.



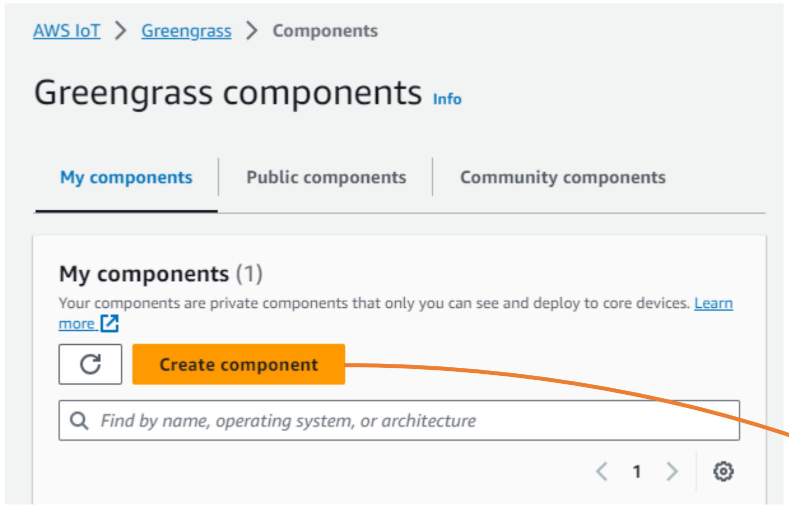
Role name
cmpe583-test-function-role-xkzsq187

Resource summary

Amazon CloudWatch Logs
3 actions, 2 resources

Step 2: Create a Custom Component

Import Lambda Function



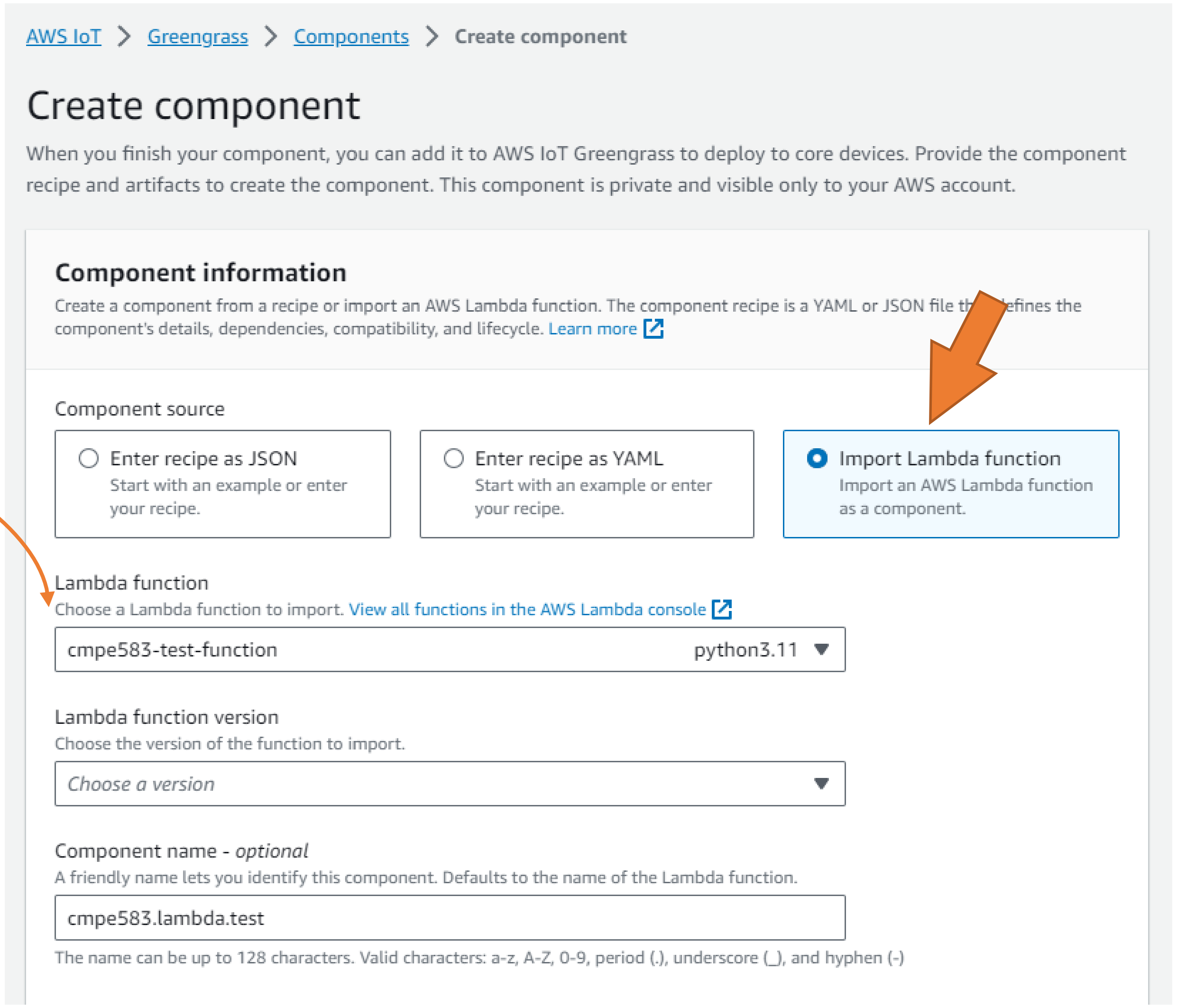
AWS IoT > Greengrass > Components

Greengrass components Info

My components | Public components | Community components

My components (1)

Your components are private components that only you can see and deploy to core devices. Learn more



AWS IoT > Greengrass > Components > Create component

Create component

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

Component information

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

Component source

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

Lambda function

Choose a Lambda function to import. View all functions in the AWS Lambda console

cmpe583-test-function python3.11

Lambda function version

Choose the version of the function to import.

Choose a version

Component name - *optional*

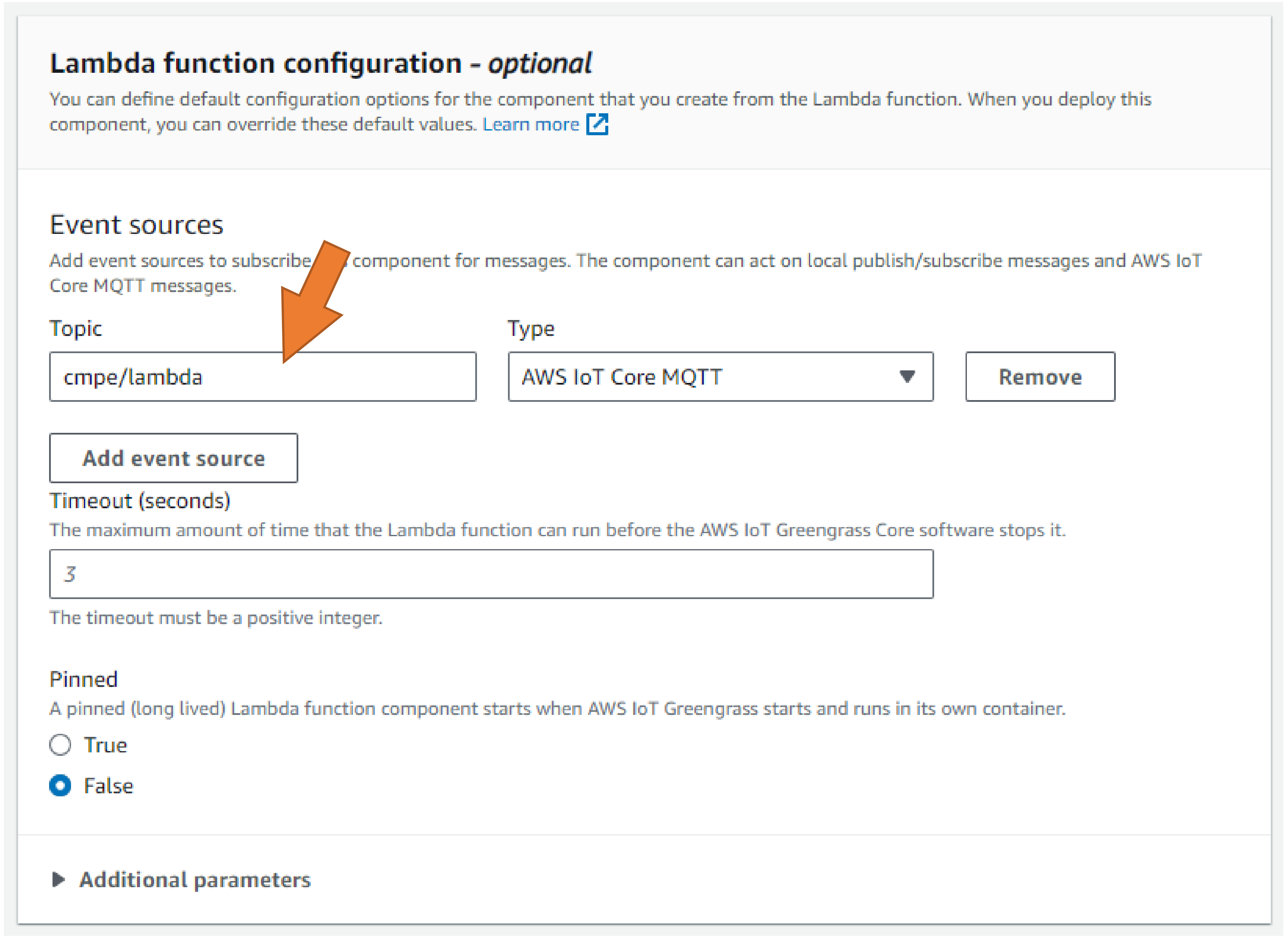
A friendly name lets you identify this component. Defaults to the name of the Lambda function.

cmpe583.lambda.test

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

Step 2: Create a Custom Component

Configure Event Source to Trigger Lambda Function



Lambda function configuration - *optional*

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

Event sources

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

Topic	Type	
cmpe/lambda	AWS IoT Core MQTT	Remove

Timeout (seconds)

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

3

The timeout must be a positive integer.

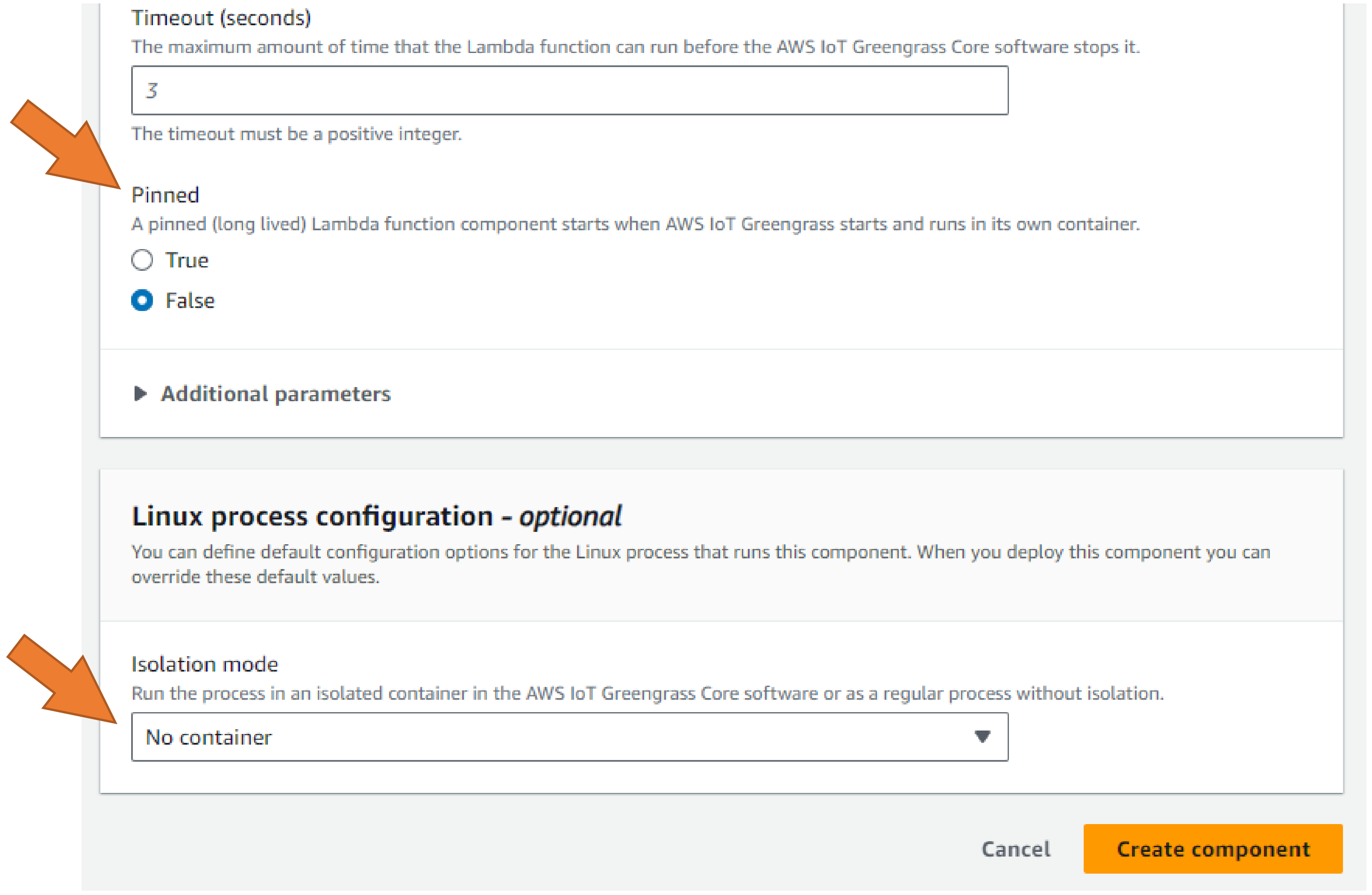
Pinned

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

True [✓ False]

Step 2: Create a Custom Component

Configure Lambda Lifecycle



Timeout (seconds)

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

3

The timeout must be a positive integer.

Pinned

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

○ True

● False

Linux process configuration - *optional*

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

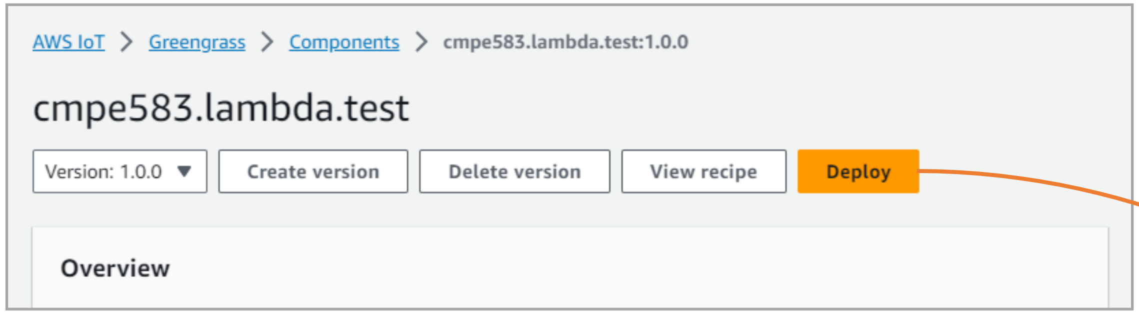
Isolation mode

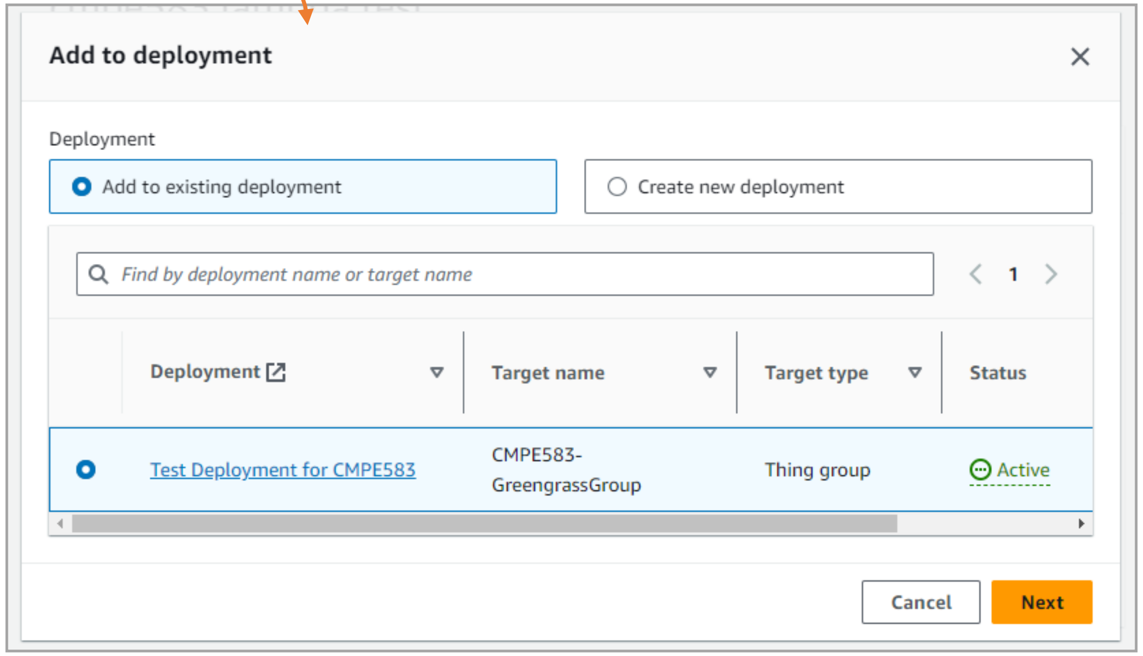
Run the process in an isolated container in the AWS IoT Greengrass Core software or as a regular process without isolation.

No container

Step 3: Deploy Your Component

Add Component to Existing Deployment





Add to deployment

Deployment

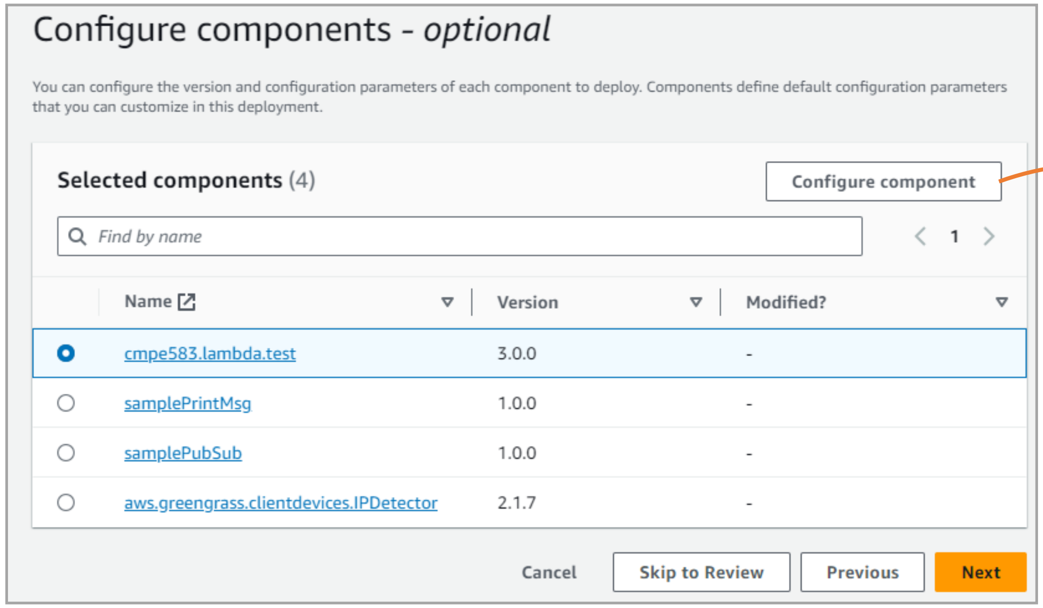
(● Add to existing deployment) ○ Create new deployment

Find by deployment name or target name

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

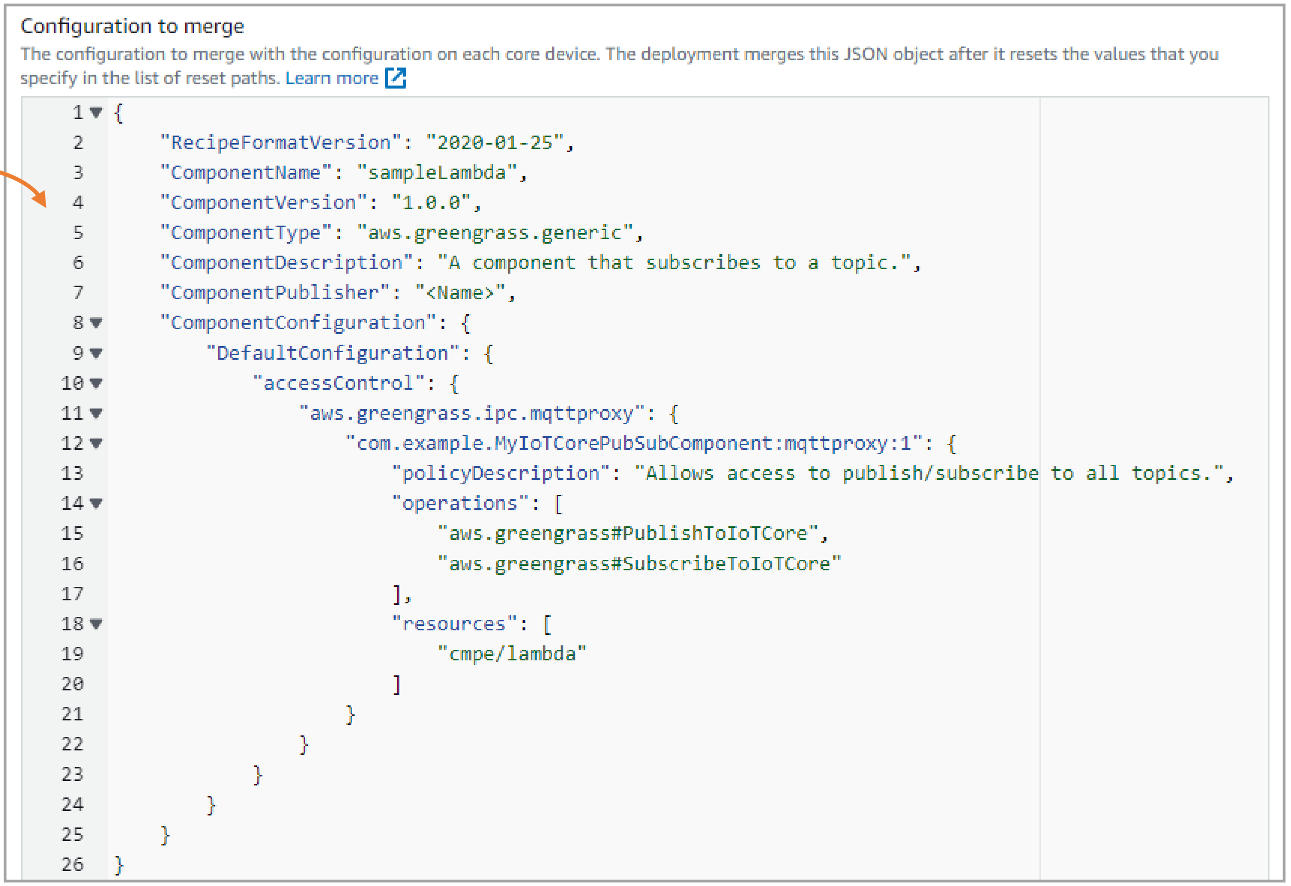
Step 3: Deploy Your Component

Configure Component to Merge Recipe



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

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

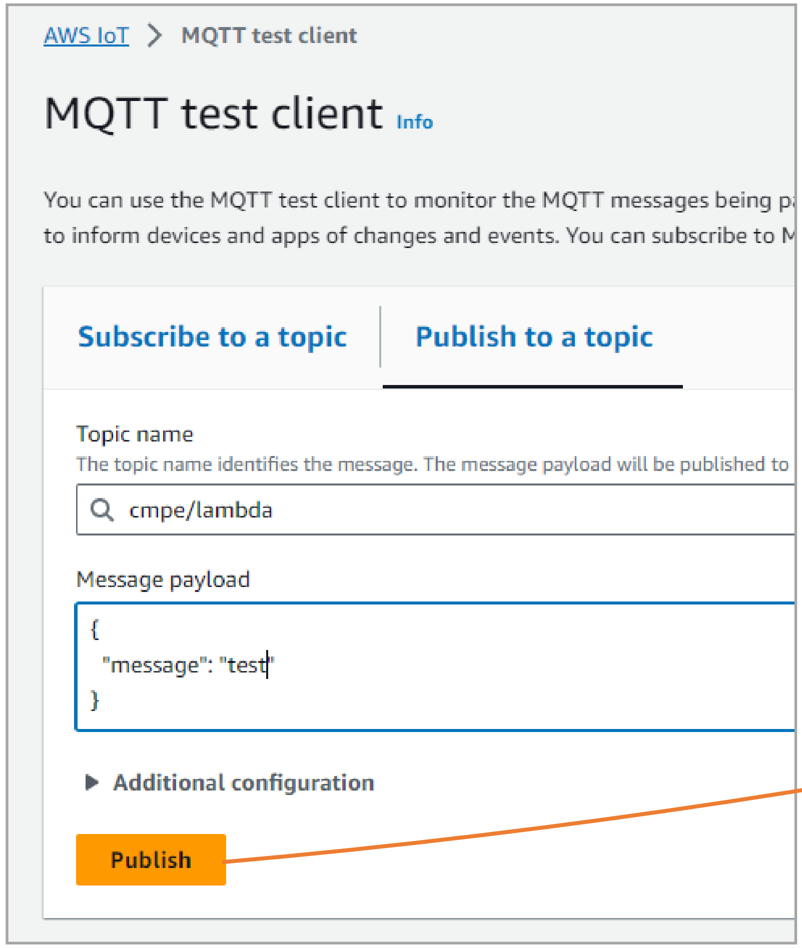


The configuration to merge with the configuration on each core device. The deployment merges this JSON object after it resets the values that you specify in the list of reset paths. Learn more

```
{
    "RecipeFormatVersion": "2020-01-25",
    "ComponentName": "sampleLambda",
    "ComponentVersion": "1.0.0",
    "ComponentType": "aws.greengrass.generic",
    "ComponentDescription": "A component that subscribes to a topic.",
    "ComponentPublisher": "<Name>",
    "ComponentConfiguration": {
        "DefaultConfiguration": {
            "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": [
                            "cmpe/lambda"
                        ]
                    }
                }
            }
        }
    }
}
```

Test Your Deployment

Via AWS MQTT Test Client



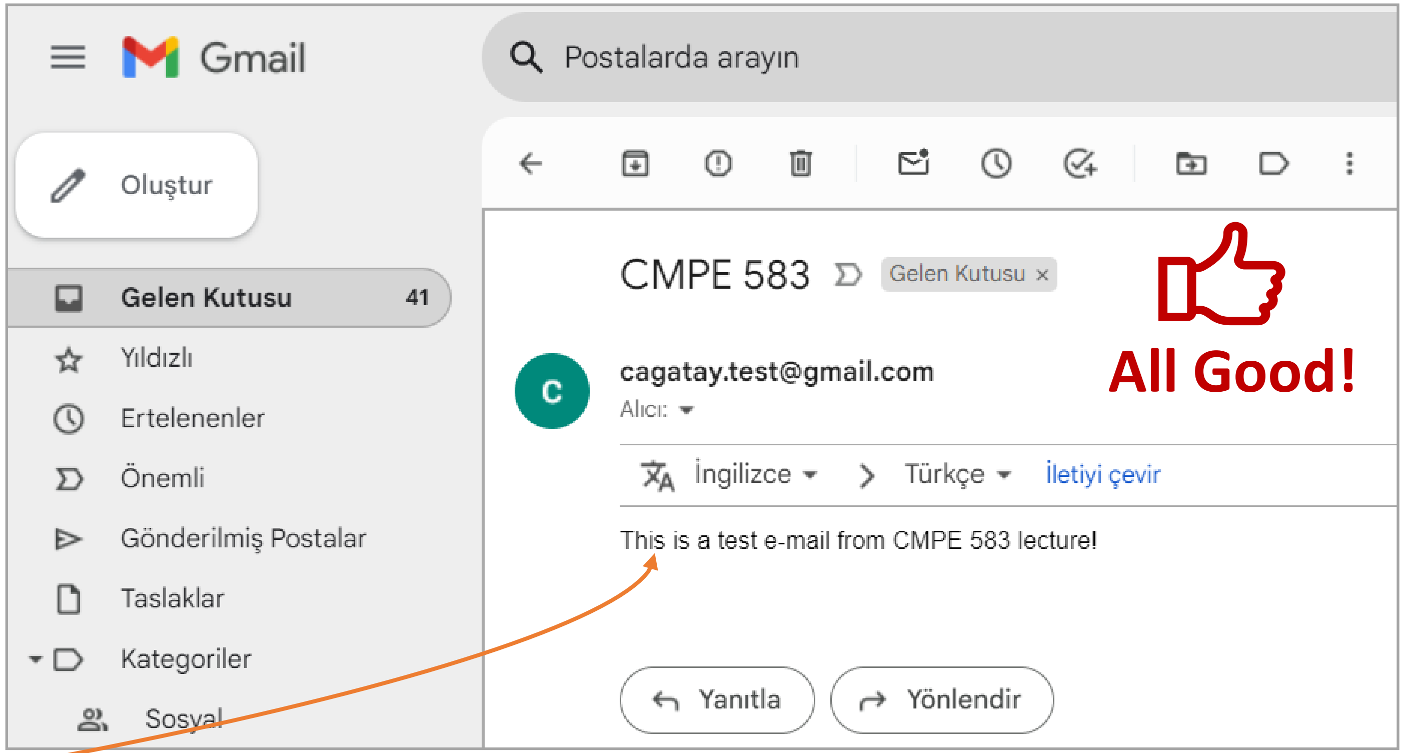
AWS IoT > MQTT test client

MQTT test client Info

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

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

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



CMPE 583

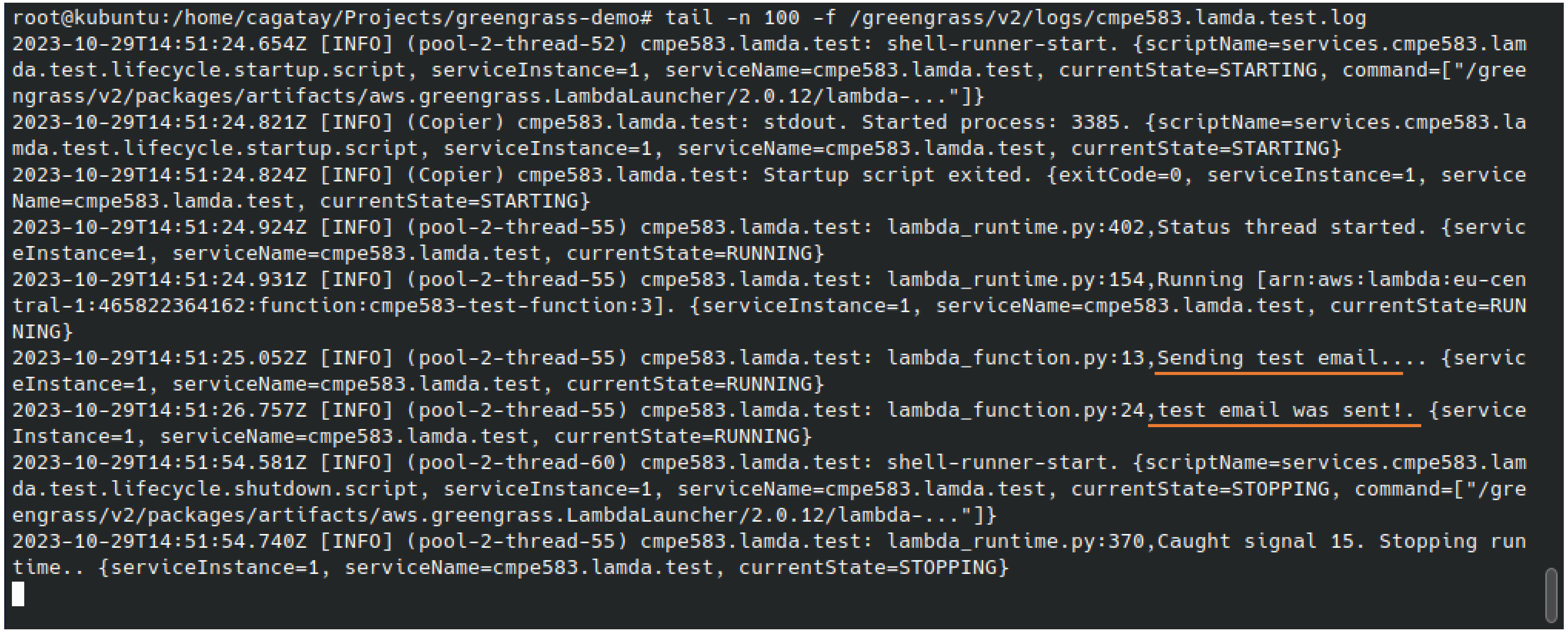
cagatay.test@gmail.com
Alıcı:

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

All Good!

Test Your Deployment

Verify From The Core Device's Logs



```
root@kubuntu:/home/cagatay/Projects/greengrass-demo# tail -n 100 -f /greengrass/v2/logs/cmpe583.lamda.test.log
2023-10-29T14:51:24.654Z [INFO] (pool-2-thread-52) cmpe583.lamda.test: shell-runner-start. {scriptName=services.cmpe583.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}
```

Thank You