Universal Just-In-Time Registration for IoT and Greengrass devices

March 20th 2019 in **Greengrass**

45 Minutes | Level 400 | Solutions Architects

The Universal Just In Time Registration block is a reference implementation of the Just In Time Registration (JITR) for the AWS platform that supports auto-registration of IoT and AWS Greengrass devices connecting themselves to AWS IoT Core using X.509 certficates signed by a custom Root Certificate of Authority (Root CA).

Scope of the Workshop

This workshop will walk you through the deployment of the Universal Just-In-Time Registration AWS Block on your account and will assist you in performing yourself the following tasks on your AWS account :

- Creating your own custom Root CA and registering it on AWS IoT.
- Creating devices certificates offline, connecting virtual devices using the created certificates to AWS IoT Core and have your device registry be automatically provisioned.
- Modifying the attributes of your device certificates to see the effects in the AWS IoT registry.

- Monitoring your registration events while new devices gets connected.

  After this workshop, you will be able to implement a JITR process to auto-register your devices at scale, leverage certificate fields to define the provisioning and the policies associated with your devices, and monitor the different registration actions happening in real-time such as provisioning successes or failures.

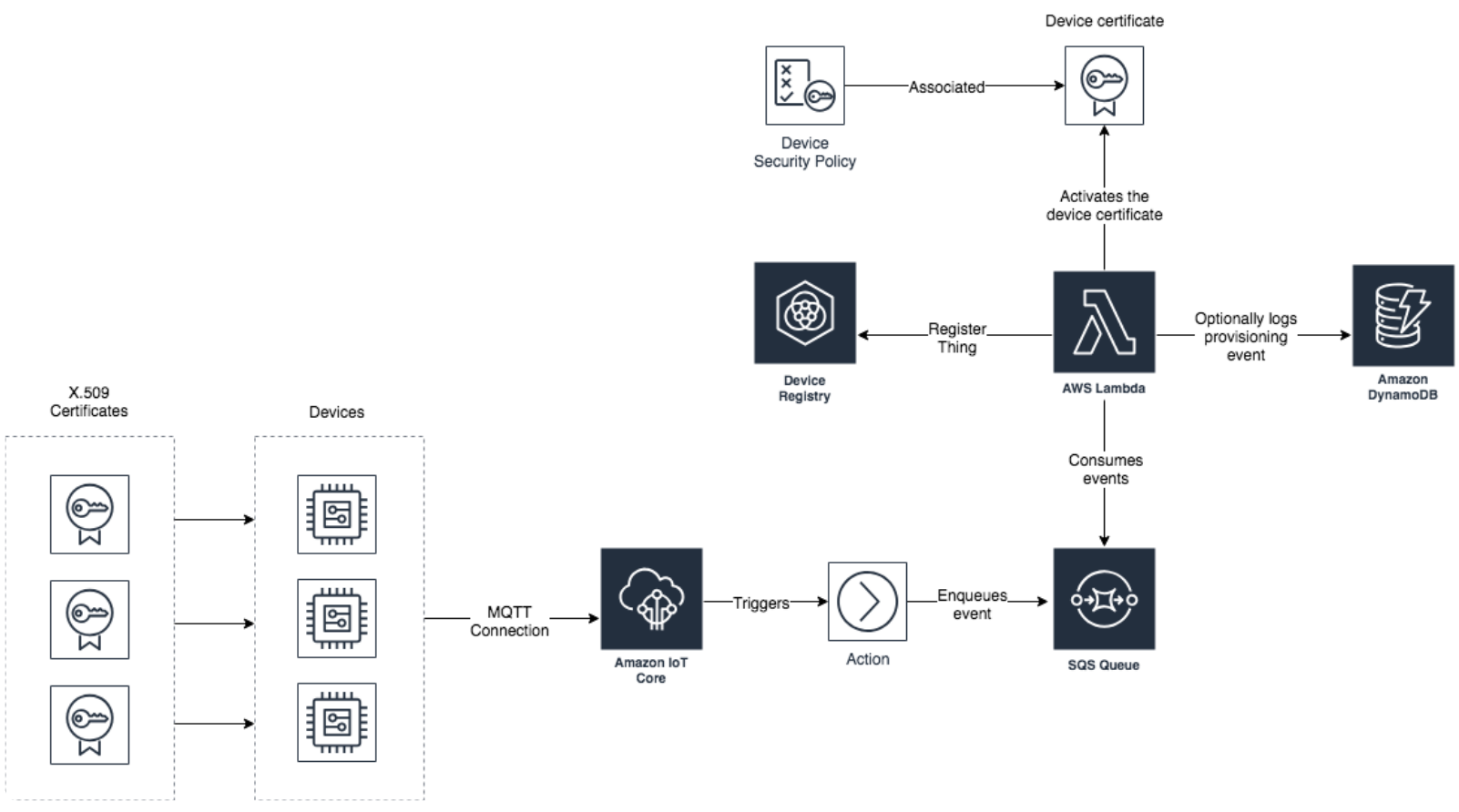
Pre-requisites

A few components are required as dependencies to this workshop before you start :

- A UNIX environment such as MacOS or Linux in order to execute the command-line tools that are part of this workshop.
- The **`openssl`** command-line tools should be installed on your operating system. These tools usually are already pre-installed on most Linux distributions, and on MacOS.
- The **`AWS CLI`** must be installed and configured with a valid AWS account.
- The **`mosquitto_pub`** command-line tools to test your generated certificates.
- You must have **`npm`** installed to install the package dependencies of the template.

Architecture Overview

The below architecture diagram details the flow associated with the Just-In-Time-Registration process we have put in place.



Installation

In this section, we will start by installing two packages on your computer by using NPM. The first one is the Universal Just-In-Time-Registration which we will require to register our things, and the second will be an NPM module associated with this workshop which contains different scripts which we will

require in the upcoming sections. To do so, please use the following commands to install both NPM modules in a local directory on your computer.

```
# Installs the Just-In-Time-Registration module.
npm install @aws-blocks/just-in-time-registration
# Installs the workshop module.
npm install @aws-blocks/jitr-workshop
```

> *You should now have a* `node_modules/@aws-blocks` *directory created which holds the code of both modules.*

Deployment

Let's first start by deploying the Universal Just-In-Time-Registration AWS Block associated with this workshop on your AWS account. To do so, you need to package the Universal JITR stack as it embeds a lambda function which requires to be uploaded to an S3 bucket first.

```
aws cloudformation package \
    --s3-bucket <an-s3-bucket-name> \
    --template-file node_modules/@aws-blocks/just-in-time-registration/cloudformation.yml \
    --output-template-file cfn.package.yml
```

The last step is to initiate the deployment of the packaged stack on your account.

```
aws cloudformation deploy \
    --stack-name just-in-time-registration \
    --template-file cfn.package.yml \
    --capabilities CAPABILITY_IAM \
    --parameter-overrides LogEventsToDynamo="true" LogEventsToCloudWatch="true"
```

> *The above command deploys the stack and sets the* `LogToDynamoDb` *and* `LogToCloudWatch` *parameters to* `true` *in order to enable registration events logging to DynamoDB and AWS CloudWatch.*

Connecting Things

In this section, we will use an automated bash script in order to create and register a new custom Root CA on your account, and connect a first device to AWS IoT which certificate has been created using the custom Root CA.

Creating a Root CA

Head using your terminal to the `certificate-scripts` directory which contains the bash scripts we will be using.

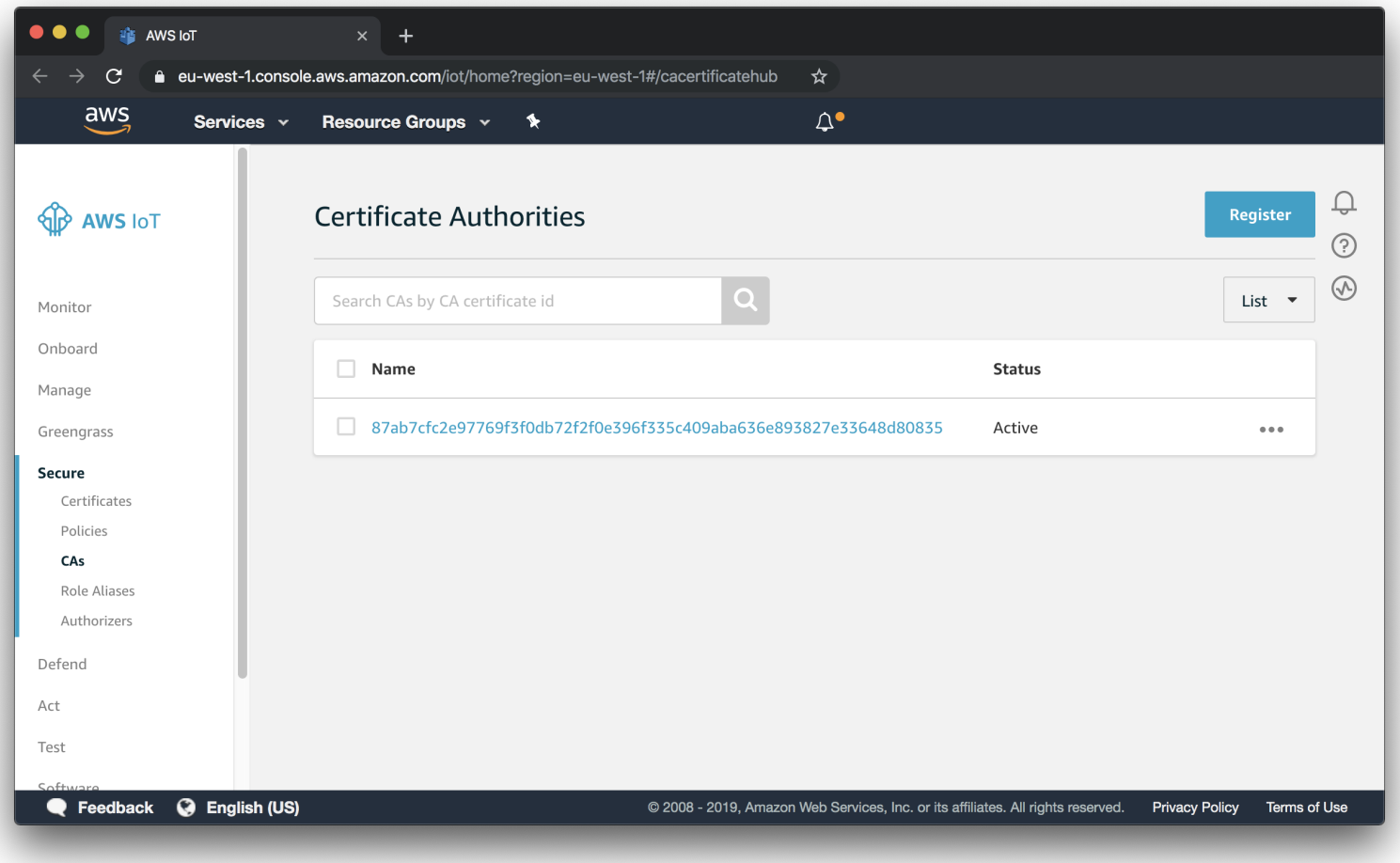
```
cd node_modules/@aws-blocks/jitr-workshop/bin/certificate-scripts
```

Next, create a new Root CA by invoking the `create-and-register-ca.sh` script.

```
# Make the script executable.
chmod +x create-and-register-ca.sh
# Create a new Root CA.
./create-and-register-ca.sh
```

> *You will be prompted to confirm whether you want to register the newly created Root CA on your account using the AWS CLI, confirm the prompt.*

Head to the AWS IoT CA console in your browser and confirm that the new CA has been registered.



Connecting your first Thing

The next step is to create a new device certificate signed using the newly created Root CA. This certificate will be created totally offline without requiring Internet or AWS access. To do so, invoke the `create-device-certificate.sh` script.

```
# Make the script executable.
chmod +x create-device-certificate.sh
# Create a new device certificate.
./create-device-certificate.sh
```

If the script was successful, you should see the newly created device certificate files in the `device-certs` directory. Once you are all set up, we'll need to try an MQTT connection to your AWS IoT Core endpoint using the new device certificate. This is where the `mosquitto_pub` command-line tool will be necessary. First, execute the following command in order to retrieve the ATS endpoint associated with your AWS IoT Core endpoint.

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

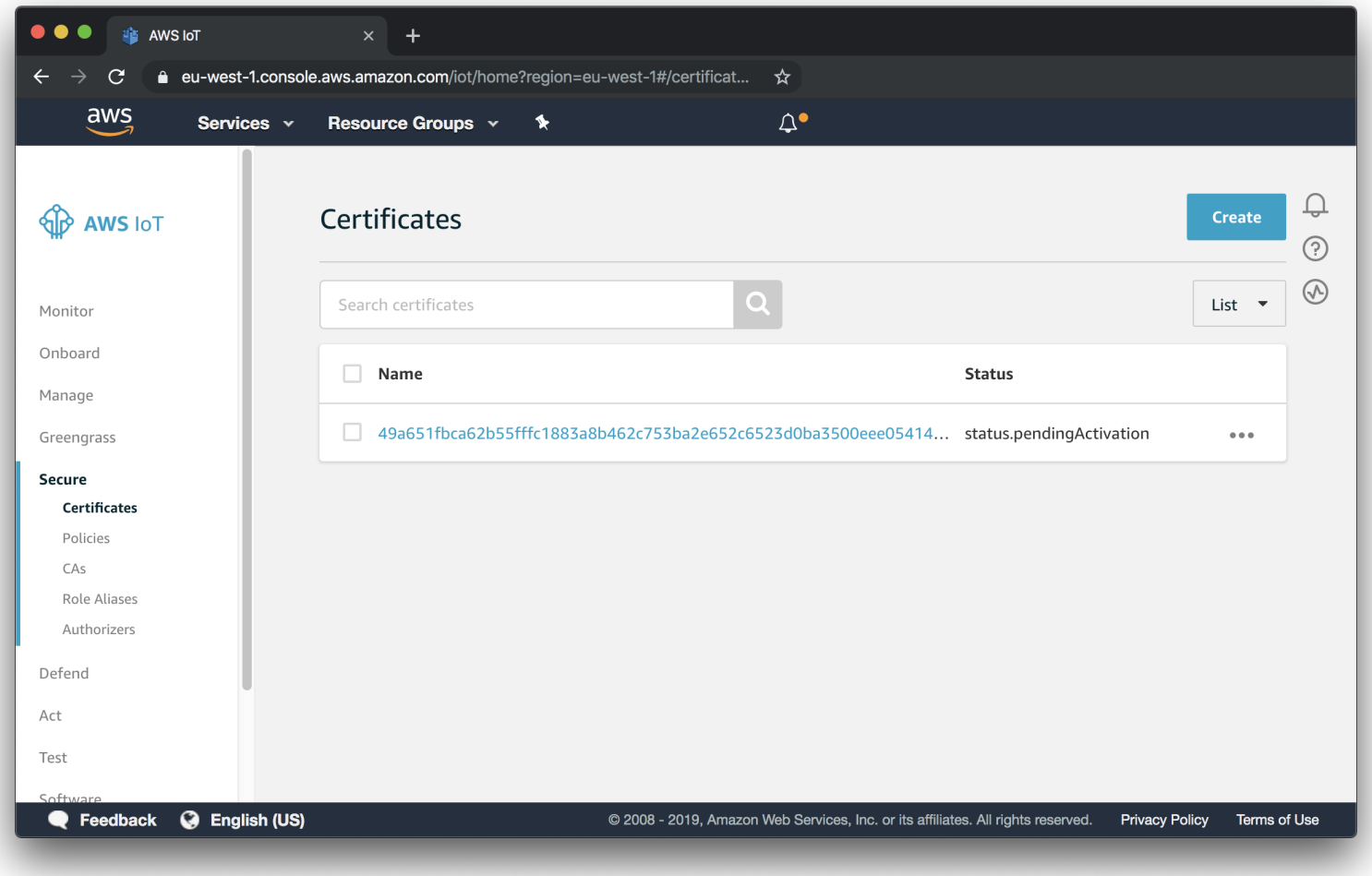
We will provide this tool all the necessary information in order for it to connect as shown in the below example which is going to :

- Connect to your AWS IoT Core ATS endpoint using the new device certificate.
- Specifies the MQTT client id as the thing name of the device in order to honor the default IoT policy of the Universal JITR stack.
- Attempts to publish a message on the `device/<thing-name>` topic which is allowed by the default IoT policy.

```
mosquitto_pub -d \
  --cafile aws-root-cert.pem \
  --cert device-certs/device-and-ca-certificate.crt \
  --key device-certs/device-certificate.key \
  -h <ats-endpoint> \
  -p 8883 \
  -t device/thing-1234-5678-9123 \
  -i thing-1234-5678-9123 \
  --tls-version tlsv1.2 \
  -m "Hello World !"
```

> *Replace the `ats-endpoint` placeholder by the endpoint you gathered from the previous step.*

After having executed this command, it will fail with an error indicating that the `Connection was lost.` This is because the certificate associated with the device is not yet activated on the AWS IoT platform. If you go to the certificate console, you will see a new certificate entry that was automatically added by AWS IoT upon the connection of the device.

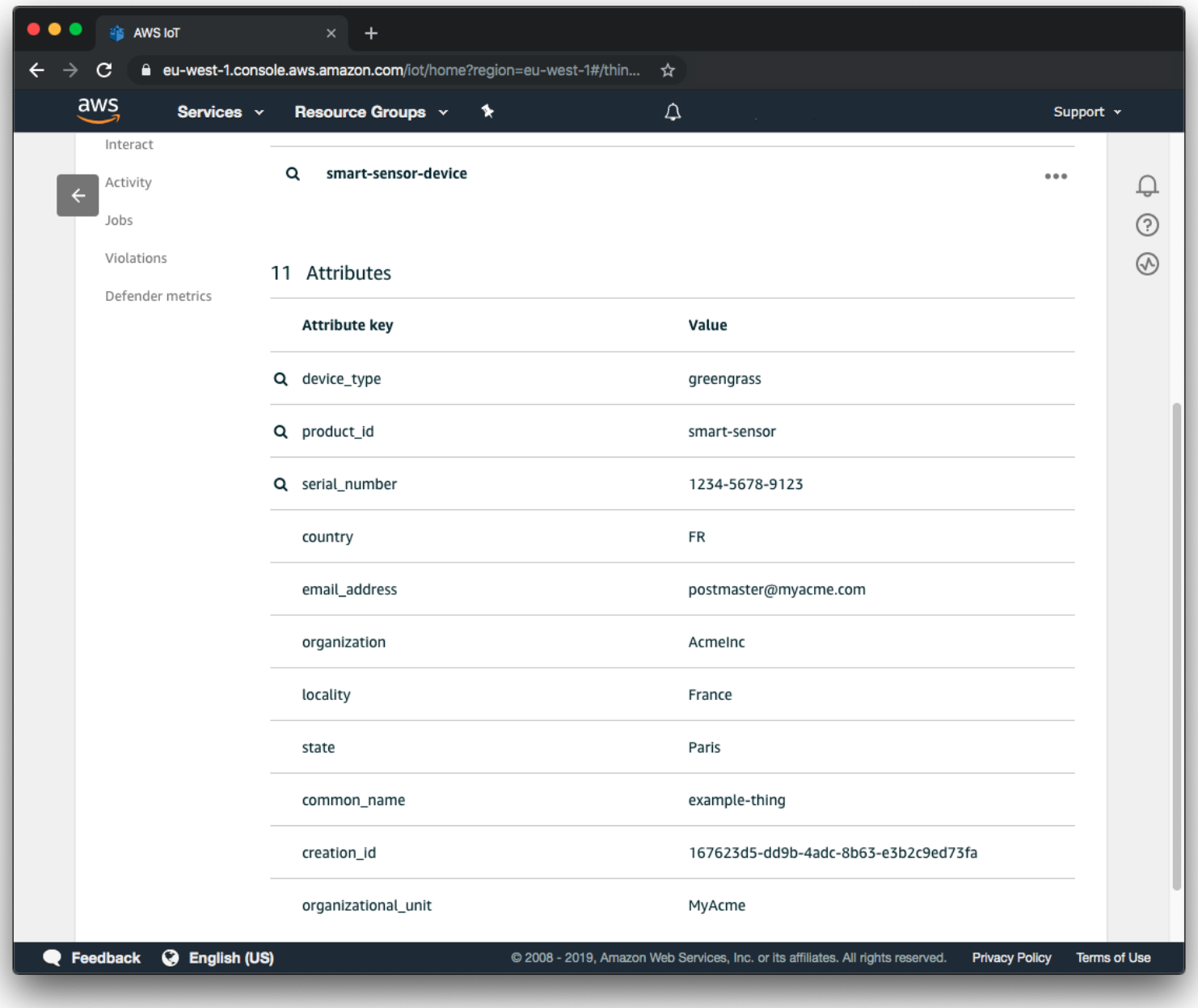


The AWS IoT platform adds a device certificate signed by a registered custom Root CA in the `status.pendingActivation` state when the device connects for the first time. The platform also invokes the lambda function associated with the Just-In-Time-Registration stack to indicate that a new device has been connected. The lambda function will then provision the device registry with a thing, associate it with a policy and if the device is a Greengrass device, will also create a new Greengrass Group before activating the certificate.

> *This means that in a Just-In-Time-Registration context, the device should retry the MQTT connection to the AWS IoT platform until it succeeds using an **exponential backoff** mechanism. When the certificate is activated by the Universal Just-In-Time-Registration block, the state of the certificate will change to `Active` .*

Viewing certificate attributes

Once the certificate transitions to the `Active` state after a few seconds, head to the Thing registry console and select the thing you just connected named `thing–1234–5678–9123`. You will see that this thing has been associated to a thing type, an IoT policy and a few thing attributes as depicted by the image below.



We can see that there are 3 searchable attributes associated with the device type (i.e `iot` or `greengrass`), the product identifier of the device, and the serial number of the device. Along these attributes, other attributes have also been set on the thing. All of these attributes are actually taken straight from the device X.509 certificate and copied in the thing attributes.

We say that the Universal Just-In-Time-Registration block is *certificate-centric* as it uses and assigns the certificate fields during the registration process. As such, every attributes defined in the certificate will be associated with the thing, making very easy to efficiently search for a thing using either the defined searchable attributes, or all of the attributes with the AWS IoT Fleet Indexing service.

Updating certificate attributes

In this section, we are going to modify the certificate generation method in order to customize the certificate attributes used by the `create-device-certificate.sh` script we used earlier. The script makes a call to the `openssl` command-line client when creating device certificates and leverages OpenSSL configuration files to define our certificate attributes.

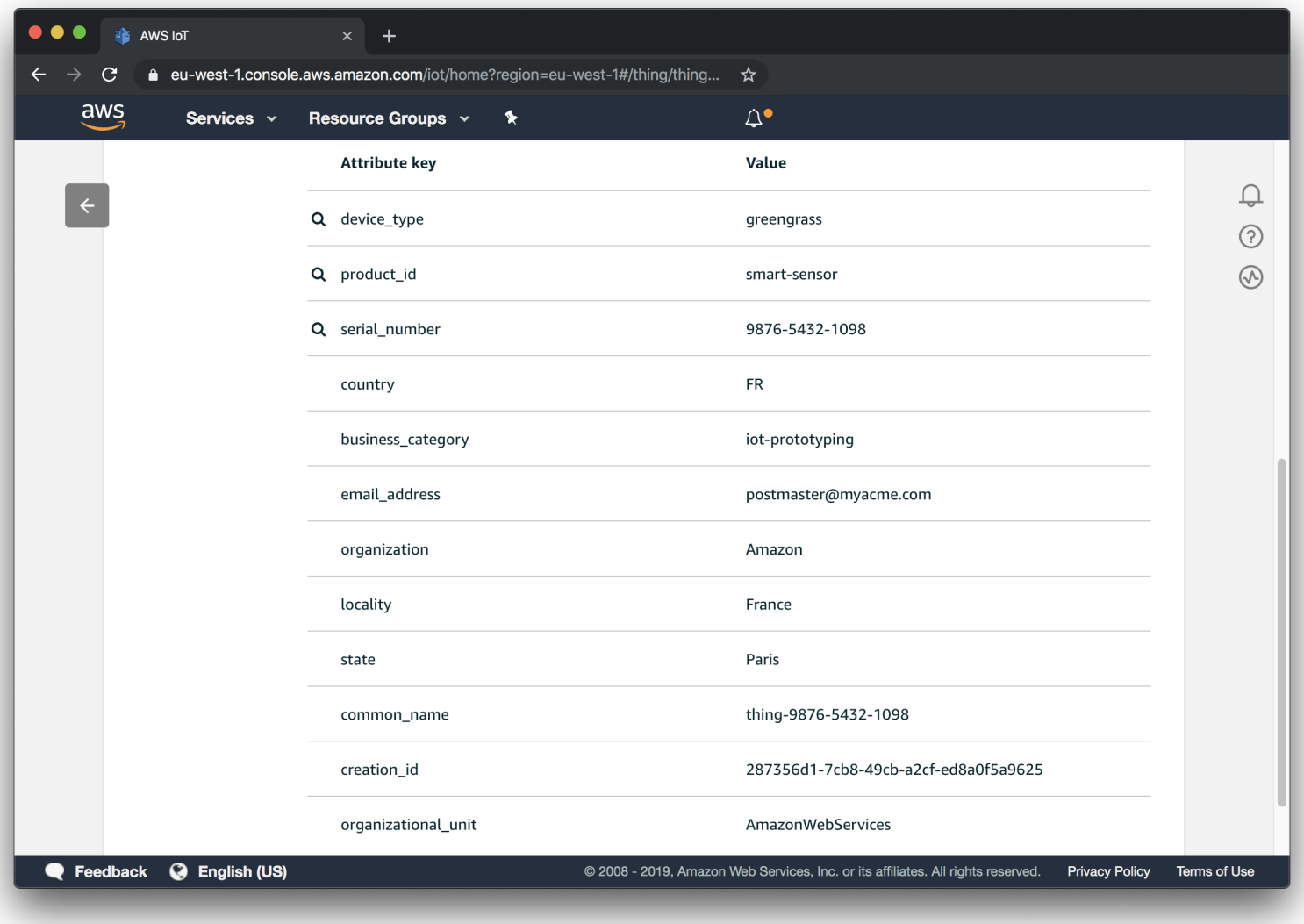
The OpenSSL configuration file used to generate device certificates is located in the workshop module in the `bin/certificate-scripts/config` directory and is named `openssl-device.conf`. Open this file using your code editor in order to make the following changes.

- Change the name of our thing defined by the `commonName` parameter to `thing-9876-5432-1098` and update the `serialNumber` to `9876-5432-1098`.
- Change the `organizationName` to `Amazon` and `organizationalUnitName` to `AmazonWebServices`.
- Change the `title` parameter (which determines the type of the device in our implementation) to `greengrass` in order for the Just-In-Time-Registration stack to consider the device as a Greengrass device.
- Finally, add a new field to the configuration file named `businessCategory` and set it to `iot-prototyping`.

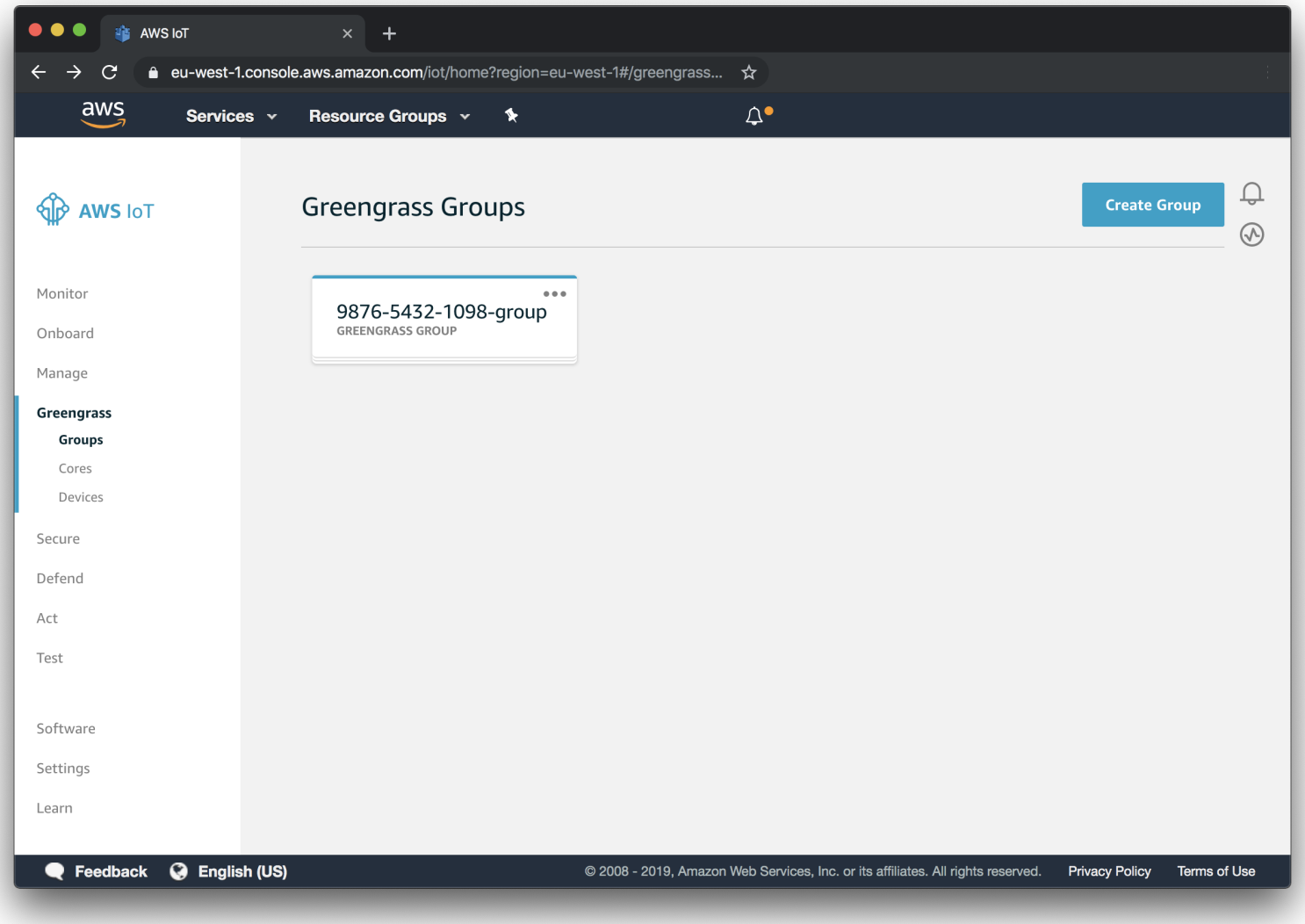
Once done, re-execute the `create-device-certificate.sh` as done earlier to generate a new device certificate using the new attributes.

```
# Create the new device certificate.
./create-device-certificate.sh
```

Re-connect the device using `mosquitto_pub` by updating the client id and the topic to match new thing name. Once the thing certificate is `Active`, head to the Thing registry console. You should see that the new thing has been provisioned in the registry and has the updated certificate fields associated to it as shown in the below image.



You will also see that since the `title` field of the device certificate has been set to `greengrass`, the Just-In-Time-Registration stack also created a ready to be deployed Greengrass group.



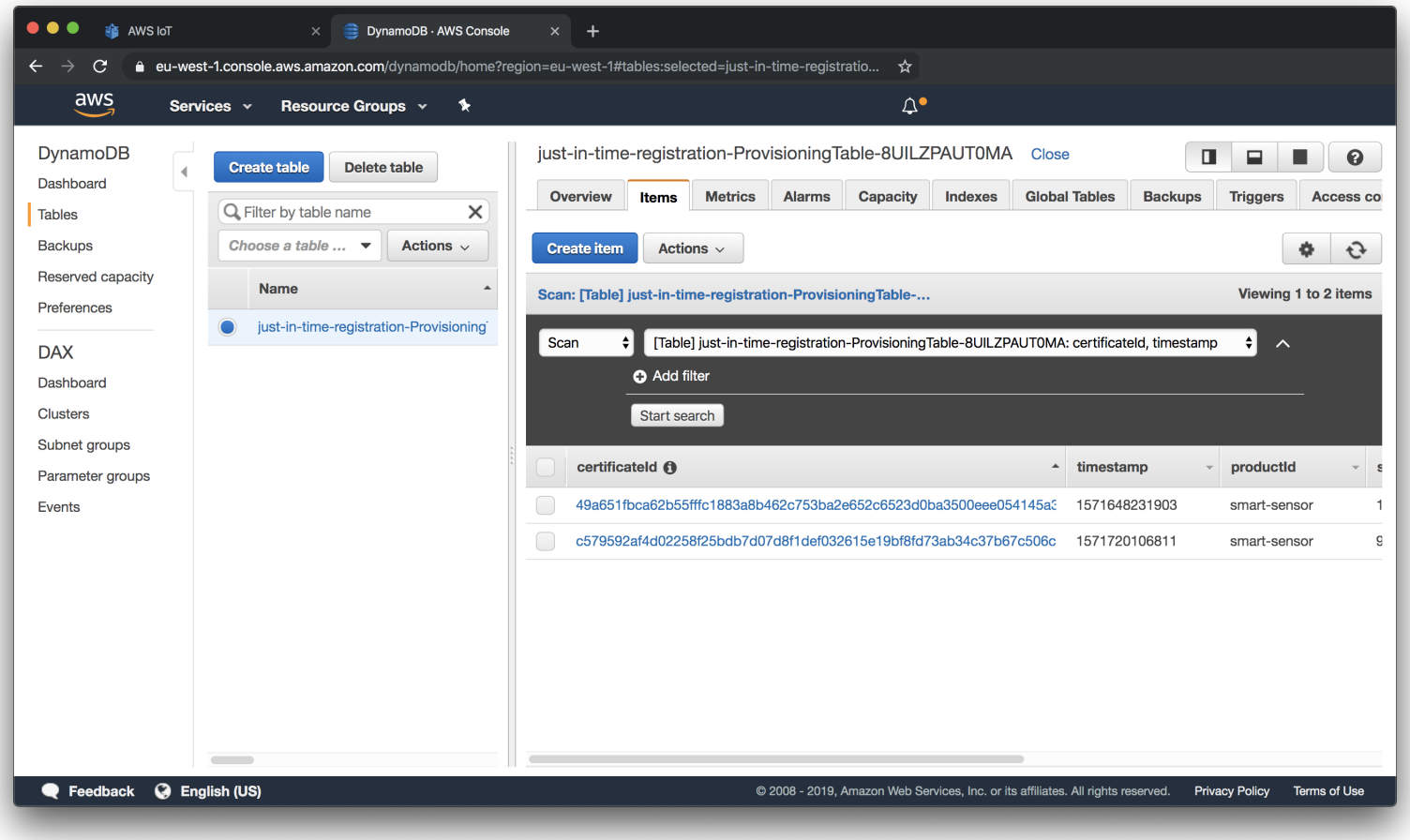
> *You can customize the way IoT Things and Greengrass Groups are named by the Universal Just-In-Time-Registration block by modifying its Cloudformation input variables. See the* ***Universal Just-In-Time-Registration documentation*** *for more information on how to update these parameters using contextual variables.*

Monitoring Registrations

A critical part of managing the operational aspect of an IoT Platform reside in the ability to monitor the behavior of devices during their lifecycle. In order to improve the way reliability engineers can monitor and assess the functionality of their Just-In-Time-Registration deployment.

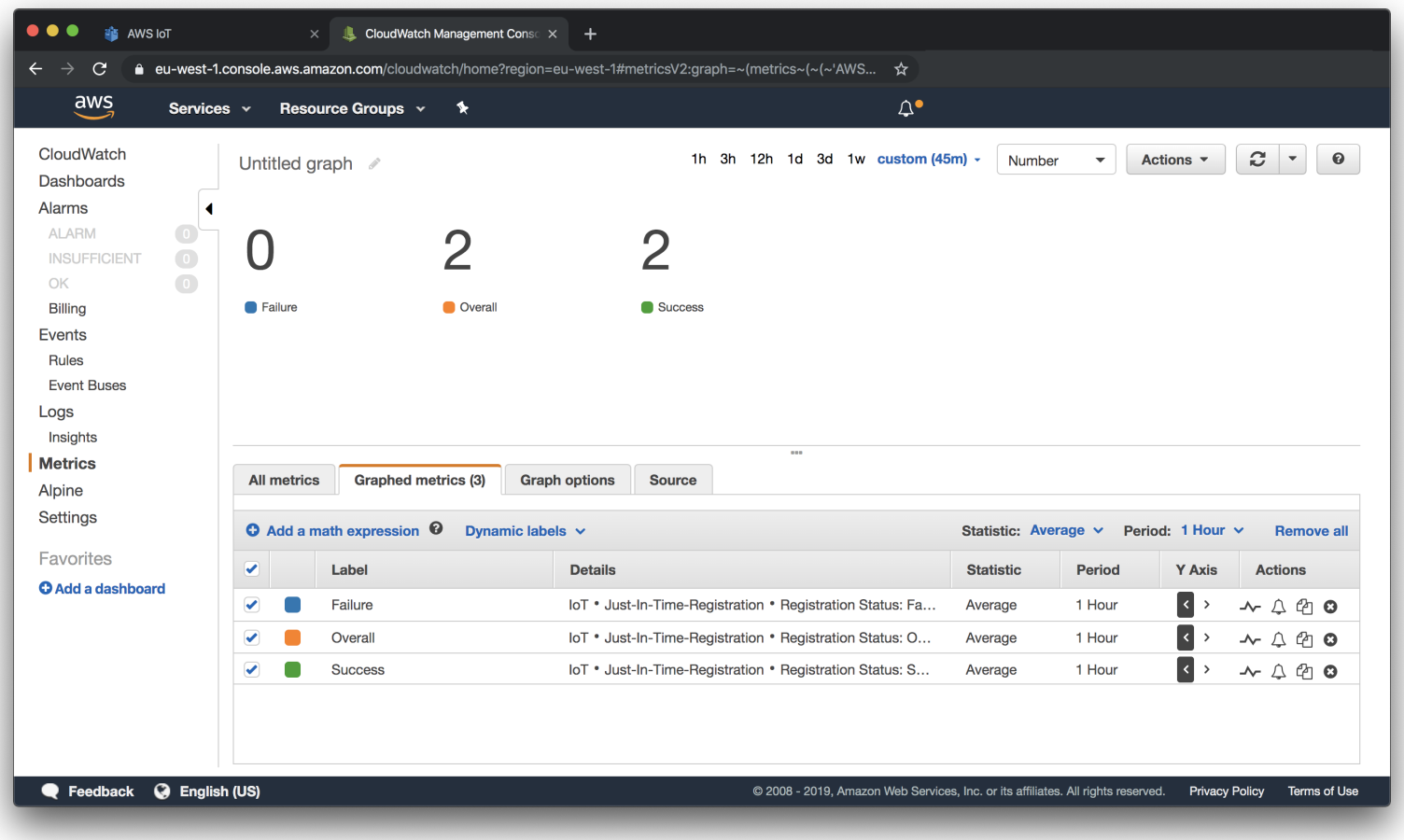
Monitoring registration logs

When deploying the Just-In-Time-Registration stack in the Installation Step, we provided the task with a `LogEventsToDynamo` variable to request the task to create a DynamoDB table and log registration events into it. In order to retrieve the registration logs associated with the two devices we connected to AWS IoT, open the DynamoDB Console and select the table associated with the stack. You will see two entries holding the details associated with the 2 registration events we triggered.



Monitoring registration statistics

We also requested the Just-In-Time-Registration stack to send statistics about the registration events to AWS CloudWatch to allow operational enginners to be able to visualize registration events and create alarms associated with these metrics. To see the metrics associated with your registrations, open the AWS CloudWatch Metrics console, and select the `IoT > Registration Status` metric.



Mass-Provisioning Simulation

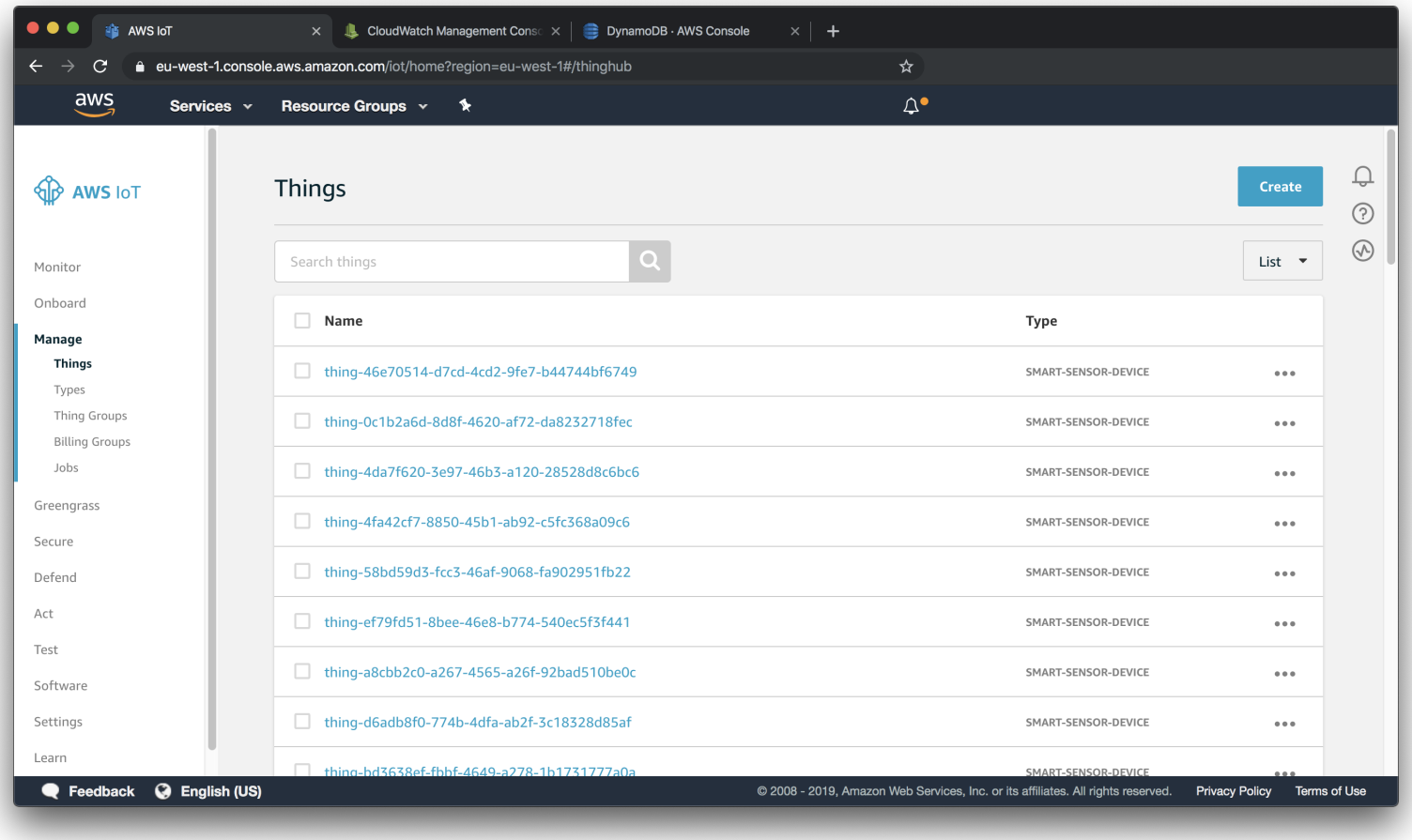
Until now we used simple bash scripts to demonstrate how to create device certificates in a step-by-step fashion. In this section, we are going to use the device simulator made available in this workshop module in order to mass provision your account with dozens of devices.

The device simulator will generate certificates on the fly with random certificate attributes using the custom Root CA we created in a previous step. To trigger the device simulator, head to its directory in the `bin/device-simulator` directory and request the simulator to create **50** new Greengrass devices using the following command.

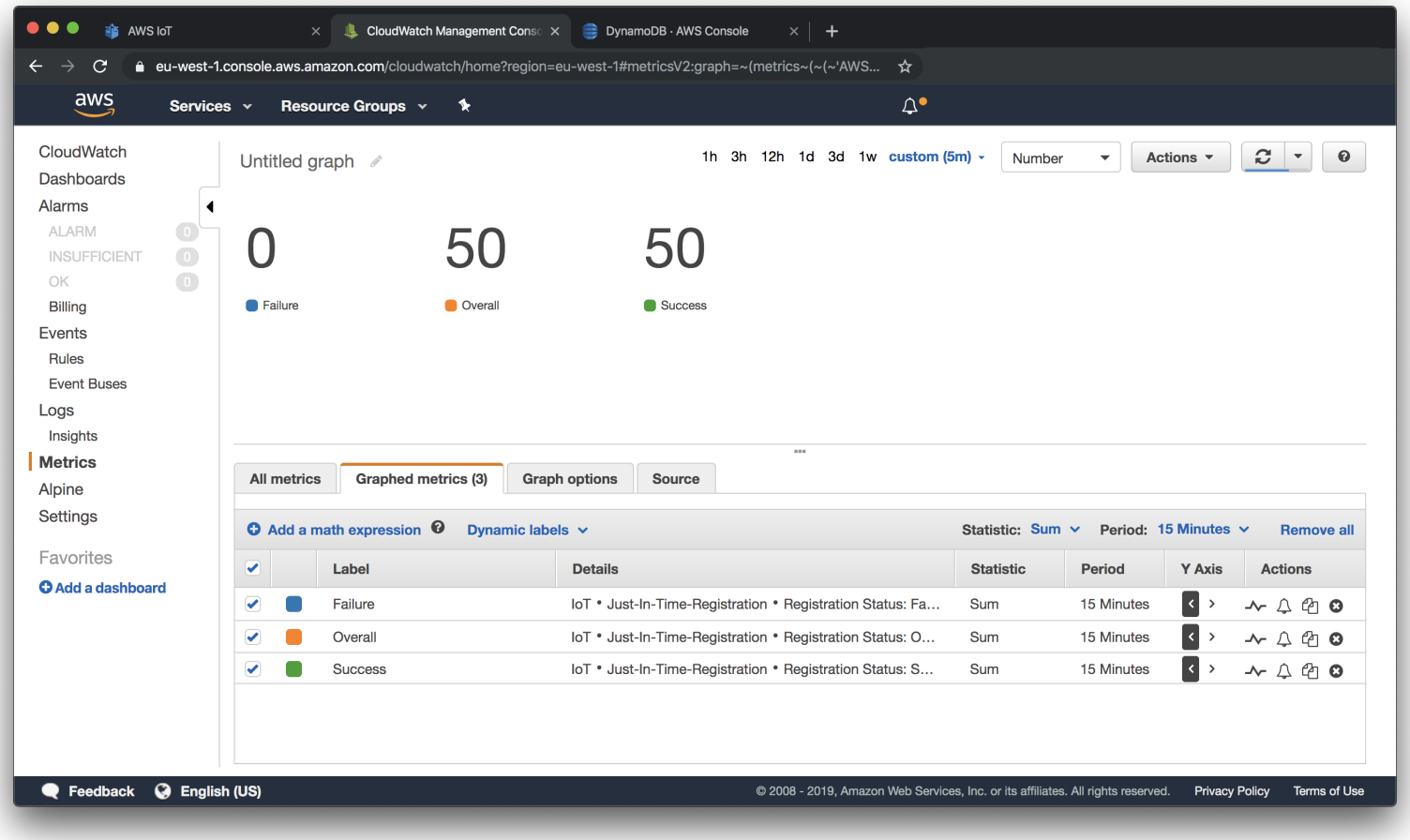
```
node index.js \
  --root-ca-directory ../certificate-scripts/root-ca-certs \
  --aws-root-ca ../certificate-scripts/aws-root-cert.pem \
  --endpoint <ats-endpoint> \
  --type greengrass \
  --number 50
```

> *Replace the `ats-endpoint` placeholder by the AWS IoT Core ATS endpoint associated with your account.*

The simulator will run until all the devices manage to successfully connect to AWS IoT Core, that is when all of the certificates have been activated. Below is a screenshot of the result in the device registry once every device managed to connect.



Browse all the Greengrass Groups that have been created on your account as well as logs from DynamoDB to get an idea of how monitoring could work for you at a larger scale.



That's all Folks, congratulations on having taken this workshop ! You know now better about the different steps involved in the Just-In-Time-Registration process and how the AWS Blocks reference implementation works. For more information, we highly recommend that you read the documentation of the Universal Just In Time Registration to understand in more details how to customize it in order to fit your needs.