Hands On - 1
Getting Started with AWS IoT

Deploying multiple **Internet of things** (**IoT**) devices on some locations and serving all requests from IoT devices needs more attention in order to obtain high availability and good performance. One of the approaches is to deploy a cloud server with high availability and advanced features. In this chapter, we will get started working with **Amazon Web Services** (**AWS**) IoT.

By the end of this chapter, you will know how to:

- Introduce AWS IoT
- Introduce IoT devices and platform for AWS IoT
- Use AWS IoT Management Console
- Use AWS IoT device SDK
- Set up AWS IoT for your IoT project
- Build a program to access AWS IoT

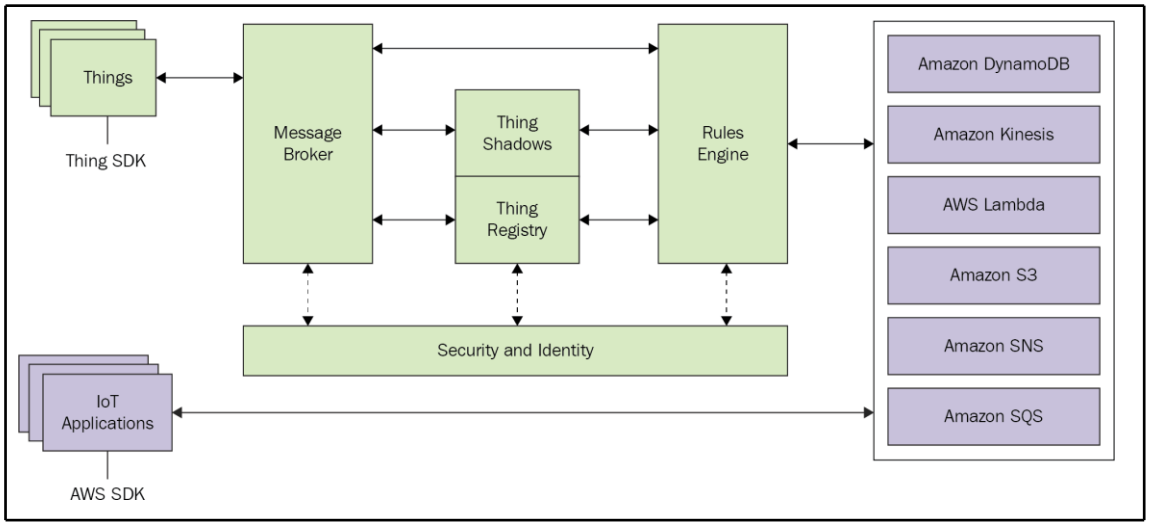
So, let's get started!

Introducing AWS IoT

In recent years, there have been a lot of IoT boards built by either manufacturers or indie makers. Each IoT offers unique features to build IoT applications to address users problems. Sensor and actuator devices are attached to these IoT boards to generate data. There is a lot of sensor data from IoT devices that we can analyze.

Suppose we have various IoT boards that are deployed on some locations. Since these IoT devices generate sensor data, we need a backend server with high availability to serve incoming data. In particular cases, we also need to analyze the data to obtain insights. To perform this scenario, we need more computing engines, such as storage and machine learning engines.

The general design of AWS IoT architecture is illustrated in the following figure. There are several components inside AWS IoT, including its endpoints. IoT devices can access AWS IoT through the AWS message broker with their own SDK. AWS IoT also provides SDK for various IoT device platforms. Using AWS IoT SDK, IoT devices can access AWS IoT directly. We will review some AWS IoT SDK, including its protocol and API, throughout this book. The AWS IoT components are shown in the following image:



From the preceding figure, we can see the following AWS IoT components:

- **Message broker**: This is basically an AWS IoT endpoint where IoT devices can access the AWS server through the **Message Queuing Telemetry Transport** (**MQTT**) protocol. Message broker also supports primitive protocols, such as the HTTP protocol. Your IoT device can send data with AWS IoT through HTTP REST.

- **Thing Registry**: This manages all the IoT device administration. You can register and configure your IoT devices, including configuring certificates and IoT device IDs.
- **Thing Shadows**: This refers to a device shadow that has functionalities to keep the current state information for a specific thing in a JSON document.
- **Rules Engine**: This provides message processing and integration with other AWS services. If you have deployed AWS services, you can apply a rule engine on those services.

Some AWS IoT components will be explored in this book.

Introducing IoT devices and platforms for AWS IoT

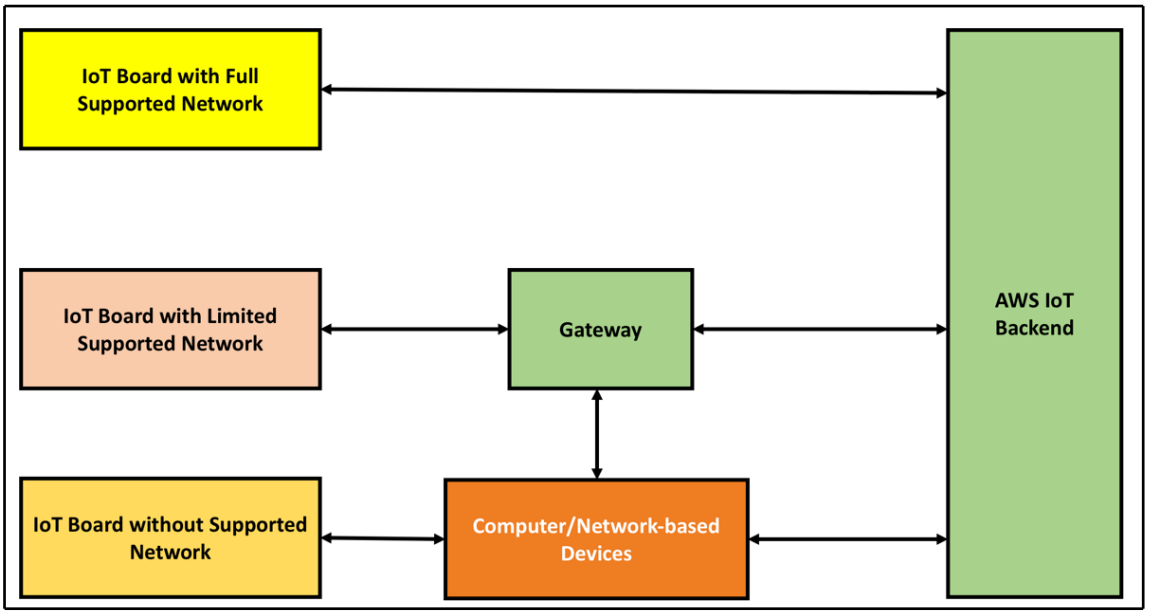
The IoT platform can connect to an internet network and interact with other platforms. Generally speaking, talking about the IoT in terms of a device platform is a huge topic. In this section, we review some IoT platforms that interact with AWS IoT.

Technically, we can describe a connectivity model between IoT devices and the AWS IoT backend. We can categorize the IoT device platform into three models based on their supported connectivity. For an IoT device with network capabilities, if this device has support for all the required AWS IoT devices, then this device can access AWS IoT directly.

Several IoT devices probably have network capabilities, but their supported protocols are not covered by AWS IoT. For this scenario, we need to build a gateway that serves and translates the IoT device protocol to the AWS IoT protocol. This gateway provides some network capabilities, such as Bluetooth, Wi-Fi, XBee, and other RF, in order to serve all exchange of data among IoT devices and AWS IoT servers.

Finally, IoT devices without network capabilities still have a chance to communicate with AWS IoT. There are two methods that we can implement for this scenario. If the IoT device can extend its functionality, we can add a network module with the supported AWS IoT protocol. Another option is to connect this IoT device to a computer. Since a computer usually has capabilities to connect to an external network, we can build a program as a bridge between the IoT device and the AWS IoT backend. The program will interact with the IoT device, for example, by sensing and actuating, and perform a data exchange with the AWS IoT backend.

All the connectivity scenarios that we have so far discussed are illustrated in the following figure:



In the following section, we will explore several IoT device platforms that are widely used on the customer side to communicate with AWS IoT. Amazon also provides a list of AWS IoT starter kits from Amazon partners at `https://aws.amazon.com/iot-platform/getting-started/#kits`. We will review some devices with the supported AWS IoT platform.

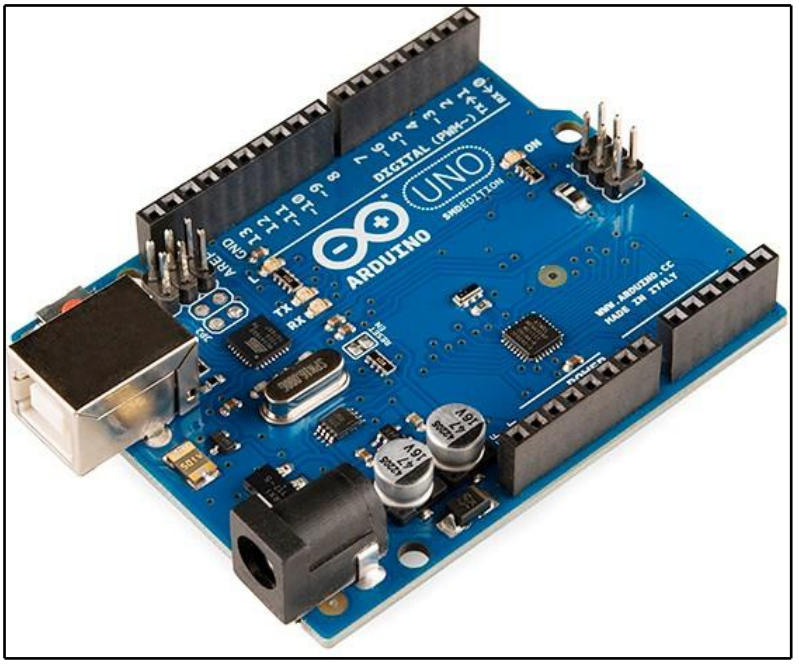
Arduino

Arduino is a widely used development board. This board is well-known in the embedded community. Mostly, Arduino boards are built using Atmel AVR, but some boards use other **Microcontroller Units** (**MCUs**) depending on who is in joint venture with Arduino. Currently, Arduino boards are built by Arduino.

We will review several Arduino boards from Arduino.cc (`https://www.arduino.cc/en/Main/Products`). We can read a comparison of all the Arduino boards by visiting `http://www.arduino.cc/en/Products/Compare`. We will review some Arduino boards, such as Arduino Uno, Arduino 101, and Arduino MKR1000:

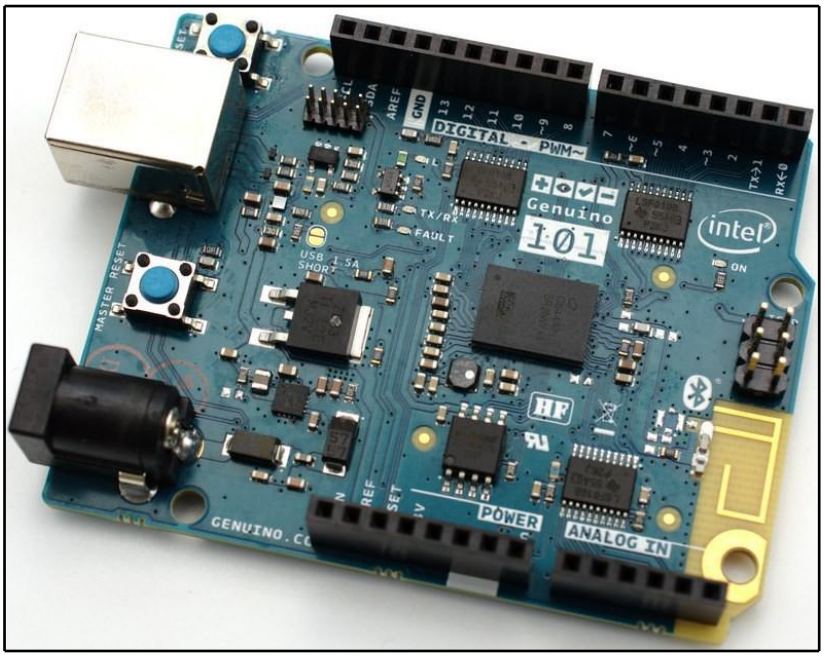
- The **Arduino Uno** model is widely used in Arduino development. It's built on top of MCU ATmega328P. The board provides several digital and analog I/O pins, which we can attach our sensor and actuator devices to. SPI and I2C protocols are also provided by Arduino Uno.

  For further information about the board, I recommend you read the board specification at `http://www.arduino.cc/en/Main/ArduinoBoardUno`. The Arduino board is shown in the following image:

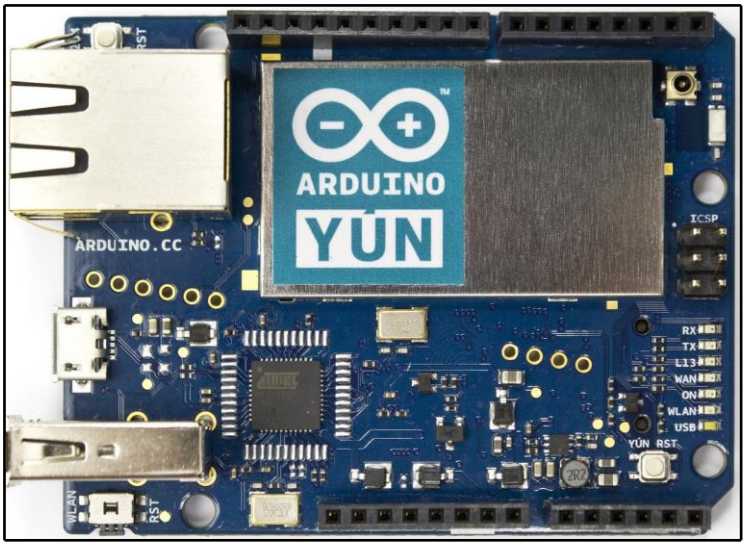


  Since Arduino Uno does not provide network modules, either Ethernet or wireless modules, we should put the network module with the supported AWS IoT to enable it to communicate with other machines.

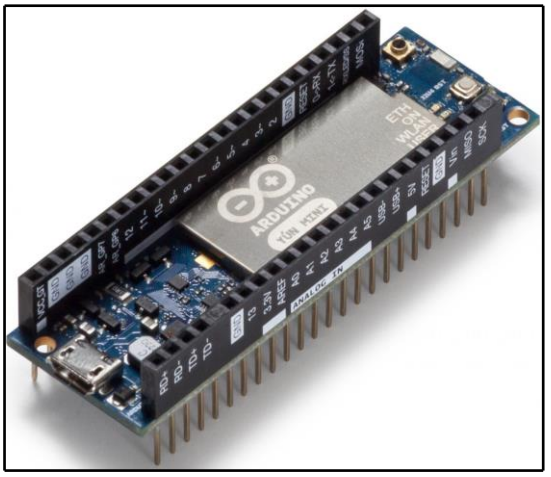
- **Arduino 101** is the same model as Arduino Uno in terms of I/O pins. Arduino 101 runs Intel® Curie™ as its core module. For more information, refer to `http://www.intel.com/content/www/us/en/wearables/wearable-soc.html`. This board has a built-in Bluetooth module. If you want Arduino 101 work with a Wi-Fi network, you should add an additional Wi-Fi shield. I recommend using Arduino Wi-Fi Shield 101. For more information, refer to `https://store.arduino.cc/genuino-101`:



- **Arduino Yún** is a microcontroller board based on the ATmega32u4 and the Atheros AR9331. This board runs OpenWrt Linux, called **LininoOS**. Arduino Yún can connect through Ethernet and Wi-Fi modules that are built-in features on the board. For further information on Arduino Yún, you can visit `https://store.arduino.cc/arduino-yun`. You can see a form of Arduino Yún in the following image:



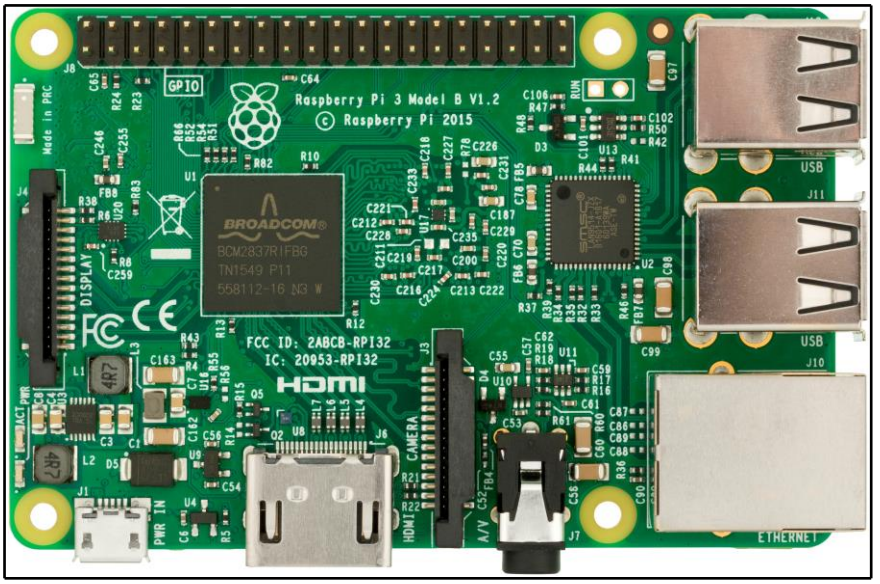
Arduino also provides another model with a small factor. It's **Arduino Yún Mini**. For more information, refer to `https://store.arduino.cc/arduino-yun-mini`. This board removes the Ethernet socket from the body to give a smaller board size. You can see Arduino Yún Mini in the following image:



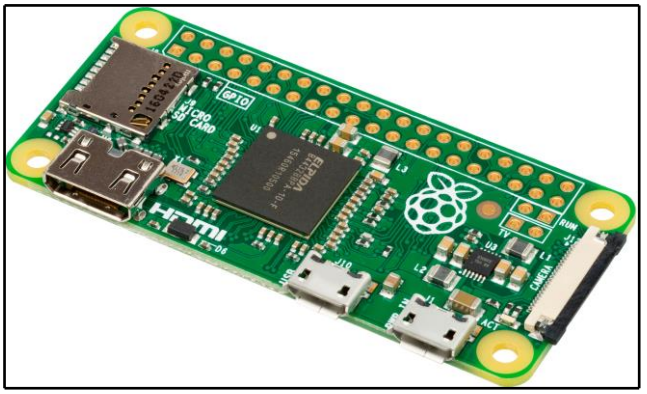
Raspberry Pi

The Raspberry Pi is a low-cost credit-card sized computer, created by Eben Upton. It's a mini computer for educational purposes. To see all Raspberry Pi models, you can refer to `https://www.raspberrypi.org/products/`. Raspberry Pi 3 Model B and Raspberry Pi Zero are described here:

- **Raspberry Pi 3 Model B**: This is the third-generation Raspberry Pi. This board consists of a Quad-Core 64-bit CPU, Wi-Fi, and Bluetooth. It's recommended for your IoT solution:



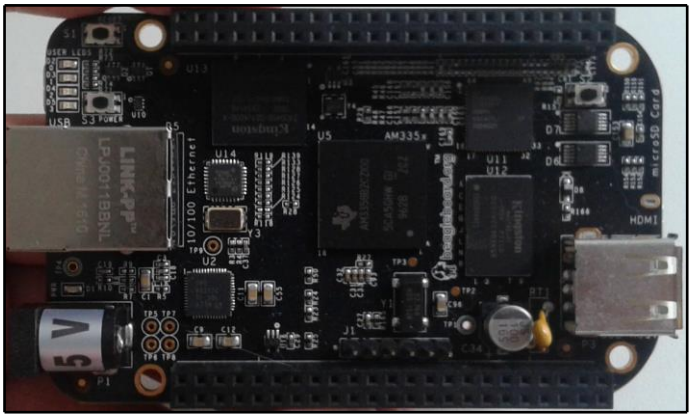
- **Raspberry Pi Zero**: This is a small computer, half the size of model A+. It runs with a single-core CPU and no network module, but it provides micro HDMI to be connected to a monitor. Since there is no network module in Raspberry Pi Zero, you can extend it by adding a module; for instance, Ethernet USB or Wi-Fi USB to connect to a network. You can see a form of Raspberry Pi Zero in the following image:



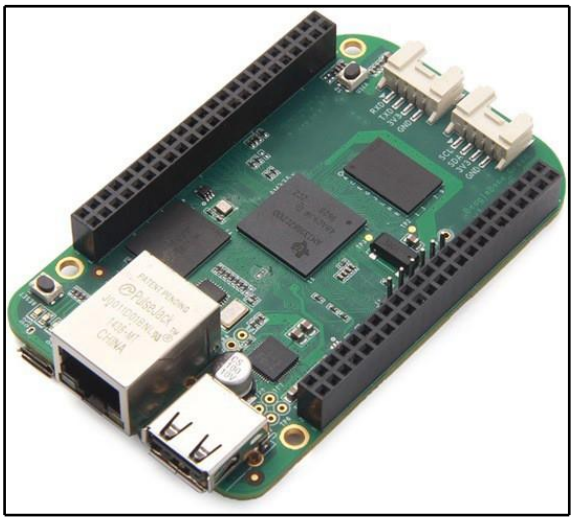
BeagleBone Black and Green

BeagleBone Black (**BBB**) Rev.C is a development kit based on an AM335x processor, which integrates an ARM Cortex™-A8 core operating at up to 1 GHz. BBB is more powerful than Raspberry Pi. The BBB board also provides internal 4 GB 8-bit eMMC onboard flash storage.

BBB supports several OS, such as Debian, Android, and Ubuntu. For more information on BBB, refer to `https://beagleboard.org/black`:



SeeedStudio **BeagleBone Green** (**BBG**) is a joint effort by BeagleBoard.org and Seeed Studio. BBG has the same features as BBB, although theHDMI port is replaced by Grove connectors, so the BBG price is lower than BBB. You can find out more and buy this board at `http://www.seeedstudio.com/depot/SeeedStudio-BeagleBone-Green-p-2504.html`:



IoT boards based on ESP8266 MCU

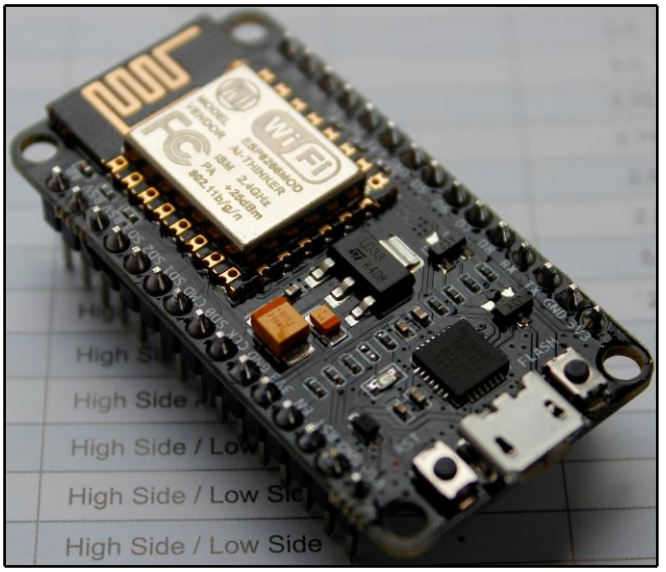
ESP8266 is a low-cost Wi-Fi MCU with full TCP/IP support. It's built by Espressif, a Chinese manufacturer. For further information about this chip, refer to `http://espressif.com/en/products/hardware/esp8266ex/overview`.

There are many boards based on the ESP8266 chip. The following is a list of board platforms built on top of ESP8266 MCU:

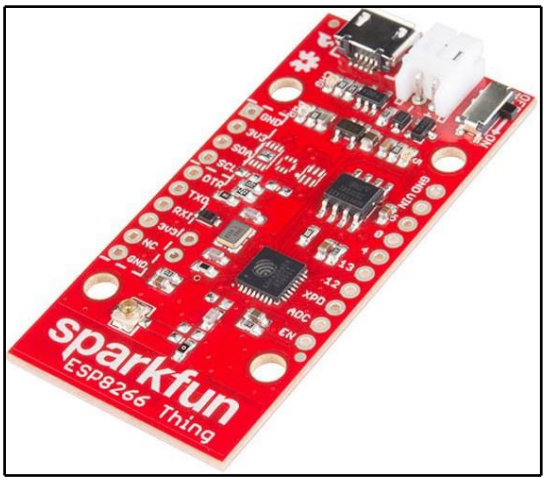
- **NodeMCU**: This board uses NodeMCU firmware, with Lua as the programming language. For more information, refer to the official website at `http://www.nodemcu.com/index_en.html`.
- **SparkFun ESP8266 Thing**: This is developed by SparkFun. You should use serial hardware, such as FTDI, to write a program in this board, but this product is ready for a LiPo charger. You can read more about it at `https://www.sparkfun.com/products/13231`.

- **SparkFun ESP8266 Thing – Dev**: This board already includes a FTDI-to-USB tool, but no LiPo charger. It's developed by SparkFun and product information can be found at `https://www.sparkfun.com/products/13711`.
- **SparkFun Blynk Board – ESP8266**: This board includes temperature and humidity sensor devices. You can read about it at `https://www.sparkfun.com/products/13794`.
- **Adafruit HUZZAH with ESP8266 WiFi**: This is developed by Adafruit. Product information can be found at `https://www.adafruit.com/products/2821`.

If you're interested in the ESP8266 chip, I recommend that you join the ESP8266 forum at `http://www.esp8266.com`.



Although NodeMCU v2 and SparkFun ESP8266 Thing boards have the same chip, their chip model is different. NodeMCU v2 uses the ESP8266 module. On the other hand, the SparkFun ESP8266 Thing board uses the ESP8266EX chip. In addition, the SparkFun ESP8266 Thing board provides a LiPo connector, which you can attach to an external battery:

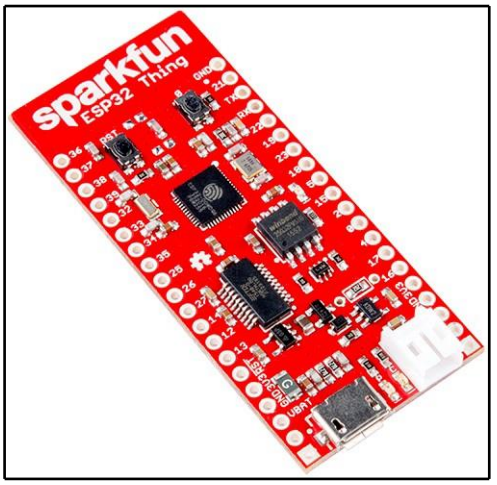


IoT boards based on ESP32

ESP32 is a chip that has two network stacks, Wi-Fi and BLE from Espressif, and is available at `http://espressif.com/en/products/hardware/esp32/overview`. This chip enables you to connect servers through a built-in Wi-Fi module. Based on my experience, there are a lot of IoT boards based on the ESP32 chip. The following is a list of ESP32 development boards:

- **SparkFun ESP32 Thing**, available at `https://www.sparkfun.com/products/13907`
- **Espressif ESP32 Development Board**, available at `https://www.adafruit.com/product/3269`

You also find various IoT boards based on the ESP32 chip at Aliexpress or online stores. A form of SparkFun ESP32 Thing is shown in the following image:



We can also use Mongoose OS ESP32-DevKitC from Cesanta to build applications for AWS IoT. To communicate with AWS IoT, they provide the Mongoose OS, which runs on ESP32. Several libraries from the Mongoose OS can be used to communicate with AWS IoT. The Mongoose OS ESP32-DevKitC from Cesanta is shown in the image source `https://mongoose-os.com/aws-iot-starter-kit/`.

IoT boards based on TI CC32XX MCU

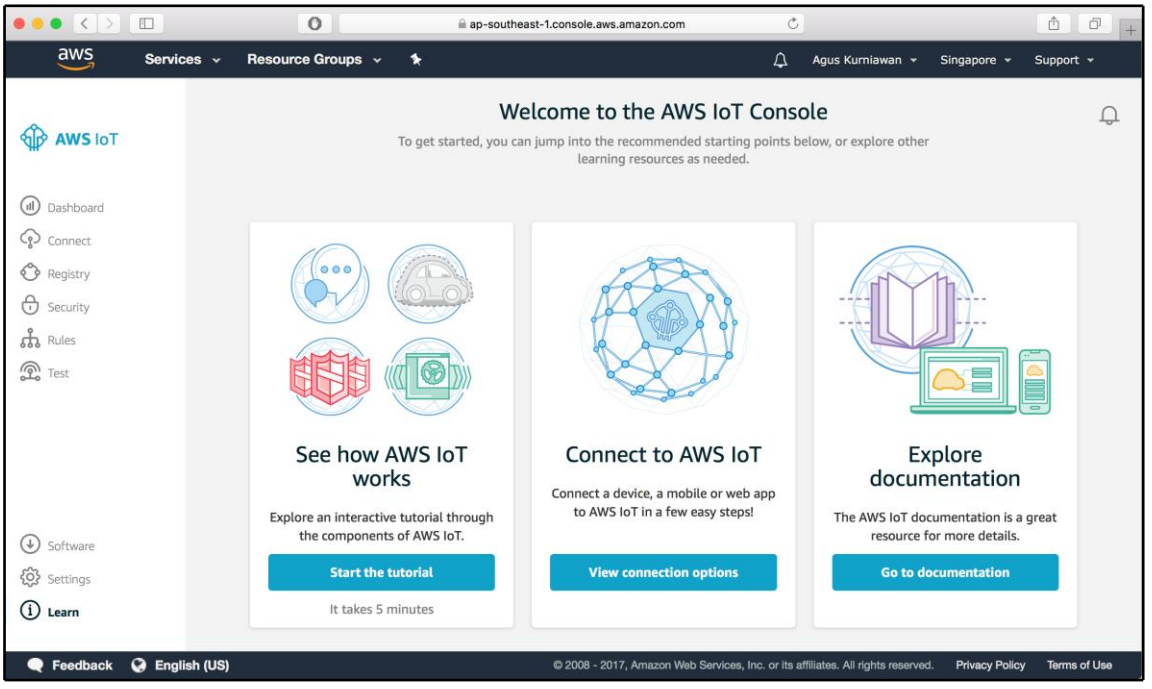
TI CC3200 is a Wi-Fi MCU from **Texas Instruments** (**TI**). The new version of TI CC3200 is TI CC3220. This chip is based on ARM Cortex-M4 from TI. This board is a complete solution for IoT. This chip is supported for station, **Access Point** (**AP**), and Wi-Fi Direct modes. Regarding security, TI CC32XX supports WPA2 personal and enterprise security and **Web Processing Service** (**WPS**) 2.0. A comparison of TI CC3200 and TI CC3220 can be found at `http://www.ti.com/product/CC3220`.

For IoT development, TI provides the SimpleLink Wi-Fi CC32XX LaunchPad evaluation kit. It's a complete kit for development and debugging. The SimpleLink Wi-Fi CC3200 LaunchPad board is shown in the website `https://www.conrad.de/de/entwicklungsboard-texas-instruments-cc3200-launchxl-1273804.html`.

TI CC3200 is also used by RedBear (`http://redbear.cc`) to develop RedBearLab CC3200 and RedBearLab Wi-Fi Micro boards. These boards have the same functionalities as the SimpleLink Wi-Fi CC3200 LaunchPad board, but exclude the CC3200 debugger tool. The price of these boards is also lower than SimpleLink Wi-Fi CC3200 LaunchPad board's price.

AWS IoT Management Console

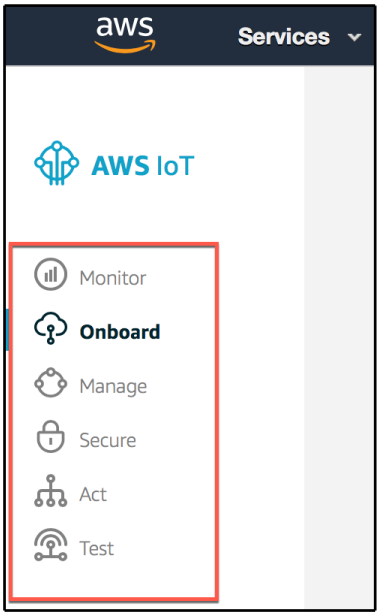
AWS IoT Management Console lets you access and manage AWS IoT through a simple and intuitive web-based user interface. This web console can be found at `https://console.aws.amazon.com/iotv2/home`. If you have an active AWS account, you should access a form of AWS IoT Management Console. A screen of AWS IoT Management Console is shown in the following screenshot:



In AWS IoT Management Console, we can manage all IoT devices. This portal provides several features, as follows:

- **Dashboard**: This shows a summary of AWS IoT usage statistics
- **Connect**: This provides information on how to connect to AWS IoT
- **Registry**: This is used to register your new IoT device or to manage existing IoT devices
- **Security**: This configures AWS IoT and IoT devices
- **Rules**: This manages all rules for AWS IoT
- **Test**: This provides a test tool to evaluate your AWS IoT platform

You can see these menus in the following screenshot:



You will probably get different menus on the dashboard. You can change your AWS region in order to get full menus by clicking the menu on the top-right of the dashboard.

We will work with AWS IoT Management Console to manage our IoT projects in the next section.

AWS IoT Device SDK

The AWS server has several components and features. To minimize complexity in development, AWS provides AWS IoT Device SDK for various IoT device platforms. You can use them directly for your IoT platform. There are a lot of objects/classes that you can apply in your IoT program to access AWS IoT. You can find out more about AWS IoT SDK at `https://aws.amazon.com/iot/sdk/`.

Currently, AWS IoT Device SDK supports the following SDK:

- Embedded C
- Arduino Yún
- Java
- JavaScript
- Python
- iOS
- Android

To work with AWS IoT Device SDK, you should verify whether your IoT device supports this SDK or not. We will focus on applying AWS IoT Device SDK in `Chapter 2`, *Connecting IoT Devices to AWS IoT Platform*.

Setting up AWS IoT for your IoT project

In this section, you will learn how to set up your IoT project, utilizing the AWS IoT platform. The following is a list of steps to build your AWS IoT project:

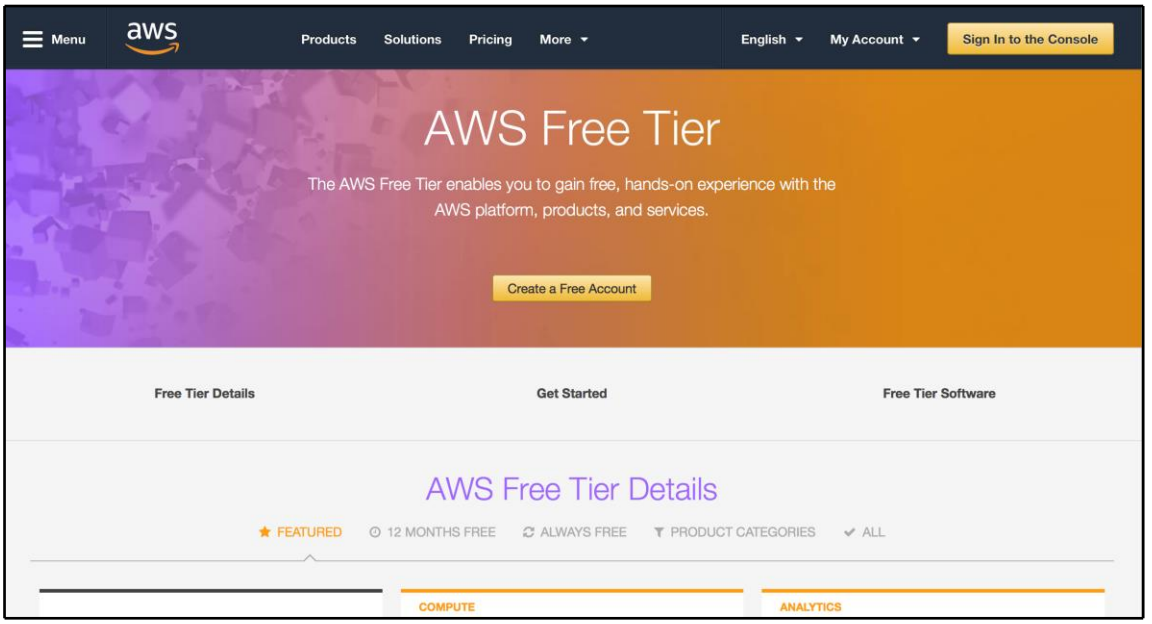
1. Register on AWS
2. Select the IoT device
3. Register AWS IoT
4. Create a security certificate
5. Configure security access

Let's go through these steps.

Creating an AWS account

AWS provides a complete solution to build your enterprise system, starting from a virtual machine and enterprise application, to machine learning and IoT. At the time of writing, Amazon offers a free one year trial access called **AWS Free Tier** for a newly registered user. You can access the full features with the limited scheme. You can register a new AWS account and get a free one year trial access at `https://aws.amazon.com`.

Most AWS can be accessed with the AWS Free Tier scheme. I recommend you do so. The AWS Free Tier registration page is shown in the following screenshot:

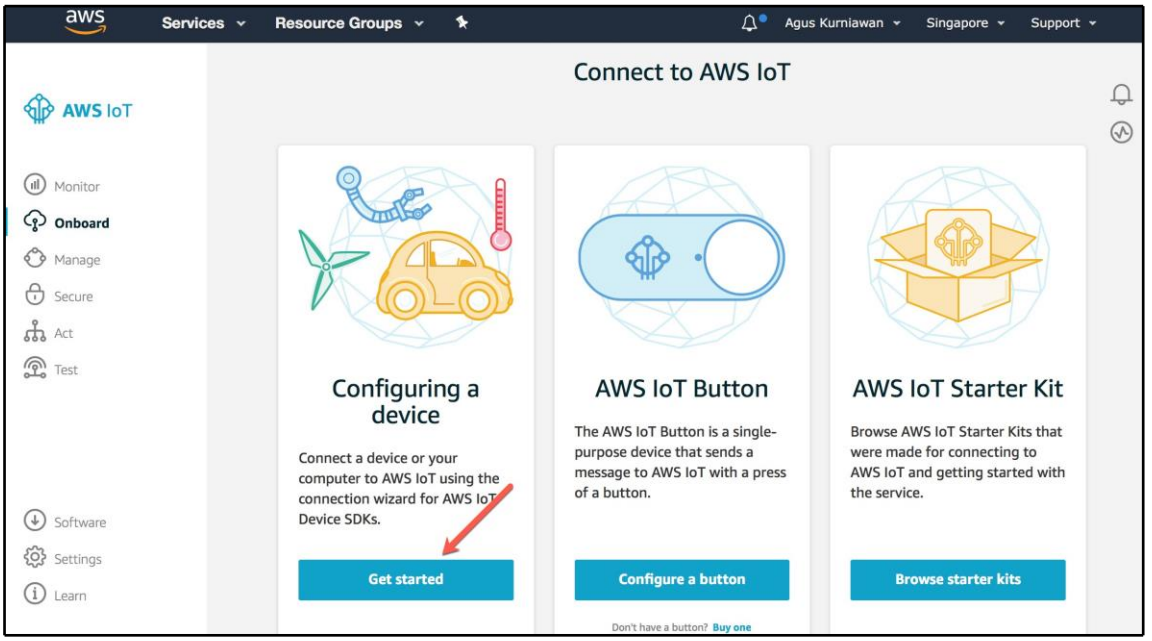


Selecting an IoT device

The next step is to select your IoT device. Each IoT device has unique capabilities. I suggest that you use the IoT device platform that is recommended by Amazon to minimize problems while developing and deploying. You can use one of the listed devices from `https://aws.amazon.com/iot-platform/getting-started/#kits`. Based on my experience, the Raspberry Pi board or IoT board with the Linux platform is easier, because most AWS IoT Device SDKs are supported.

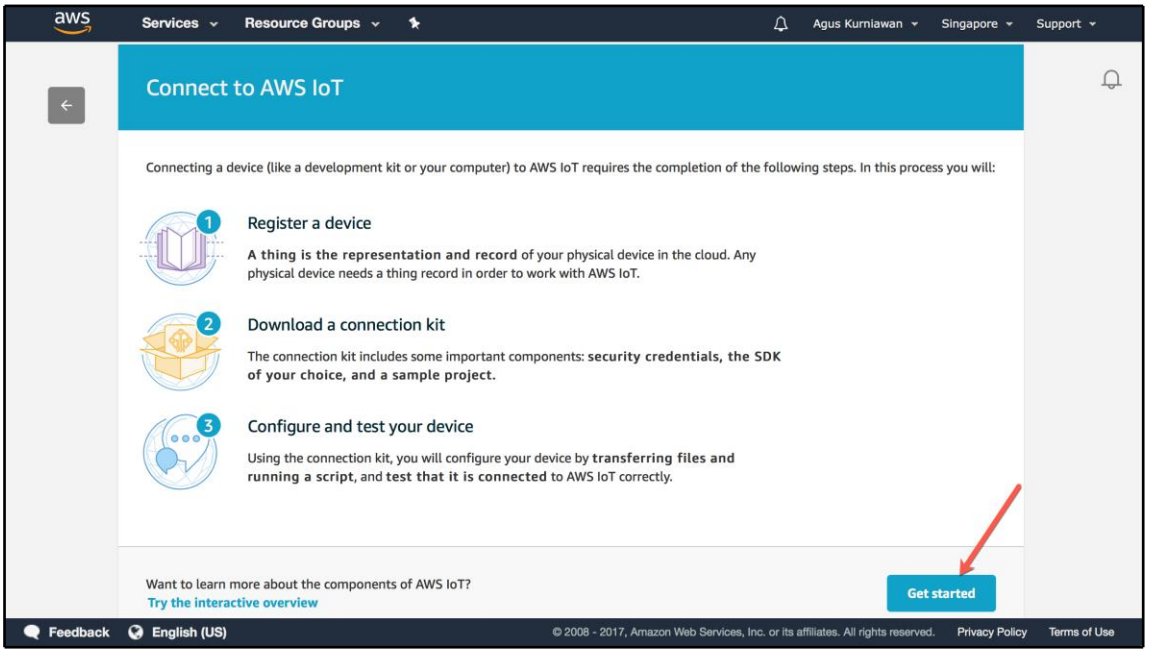
I will show how various IoT device platforms access AWS IoT, with specific scenarios in this book. Register an IoT device for AWS IoT after you have decided what IoT device model is to be implemented. You should register it in order to obtain access rights in AWS IoT. You can register your IoT device on AWS IoT Management Console with the following steps:

1. Navigate to `https://console.aws.amazon.com/iot`. You should see a form as shown in the following screenshot:

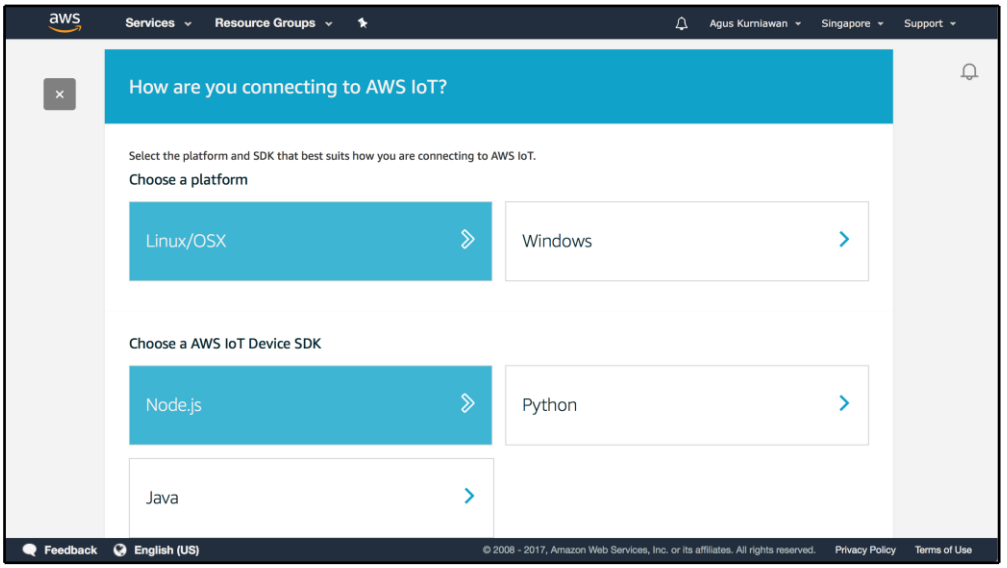


2. Select the **Onboard** option from the left-hand menu. You can click on the **Get started** button within the **Configuring a device** section, which is shown by an arrow in the preceding screenshot.

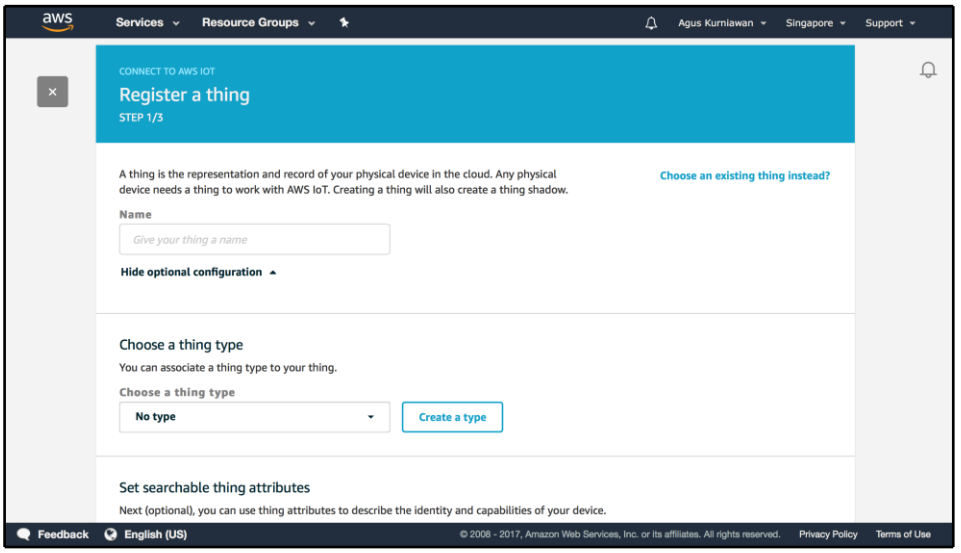
   Then, you will get information about connecting IoT device to AWS IoT, as shown in the following screenshot:



3. Select the development platform of the IoT device and AWS IoT SDK. In this scenario, I use **Linux/OSX** with **Node.js** for AWS IoT SDK:

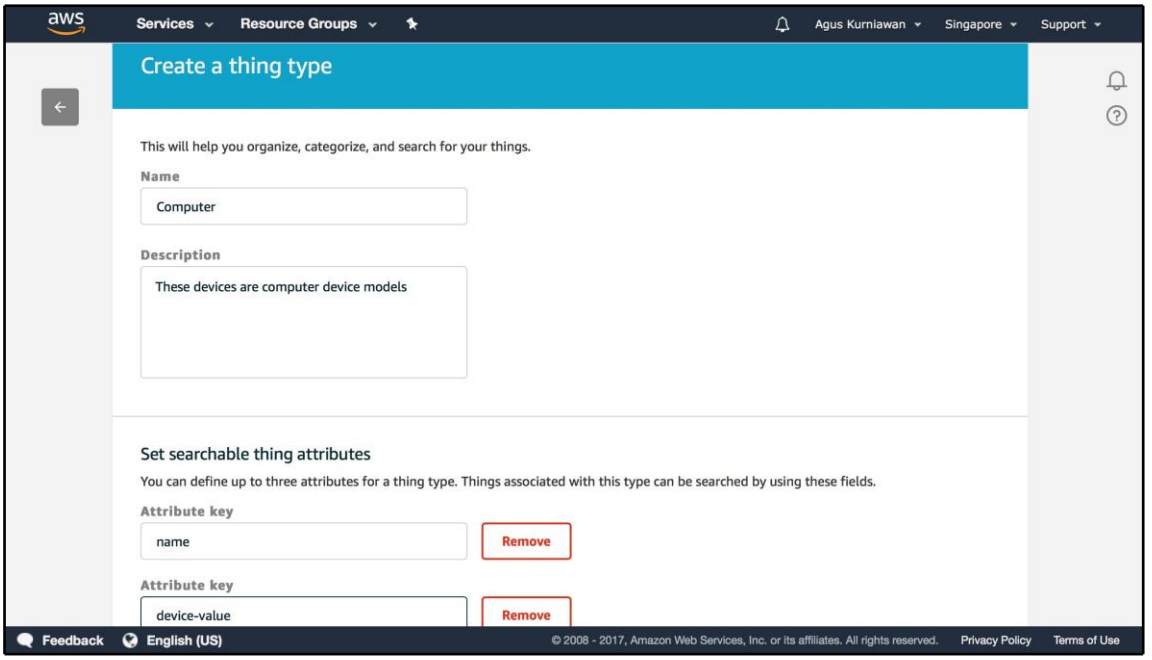


4. Now we create our IoT device name. You should define the IoT device type. To do so, you click on the **Create a type** button:

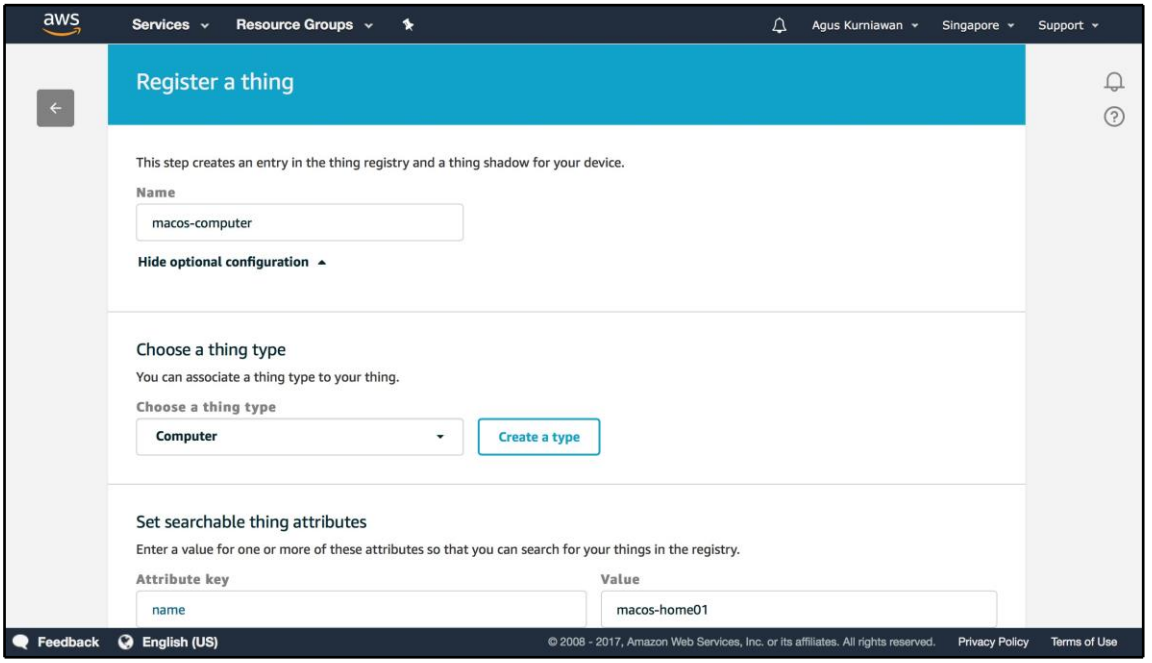


5. Fill out the IoT device type and its description. You may define IoT device attributes. For a demo, we define the following two attributes as shown in the following screenshot:
    - `name`
    - `device-value`

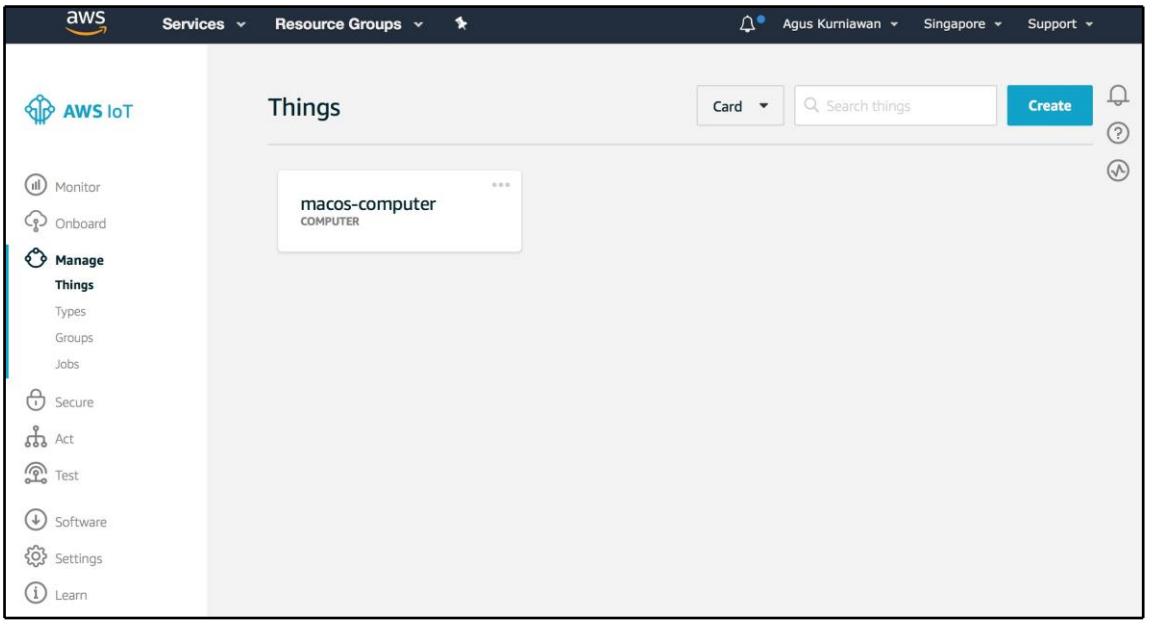
   If done, save this IoT device type.



6. Then, go back to your IoT device registering. Fill out the IoT device name and its type. I filled `macos-computer` in the **Name** field, as shown in the following screenshot:



7. If done, you should see your IoT device on the **Manage** | **Things** menu:

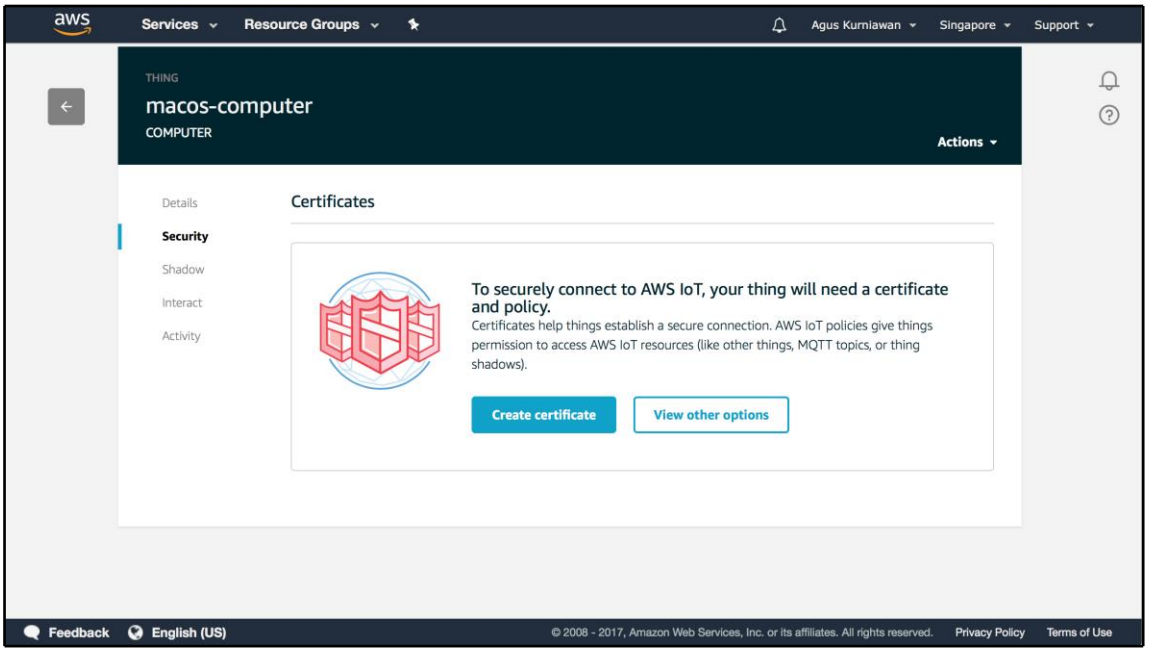


You can add additional IoT devices to simulate the AWS IoT scenario.

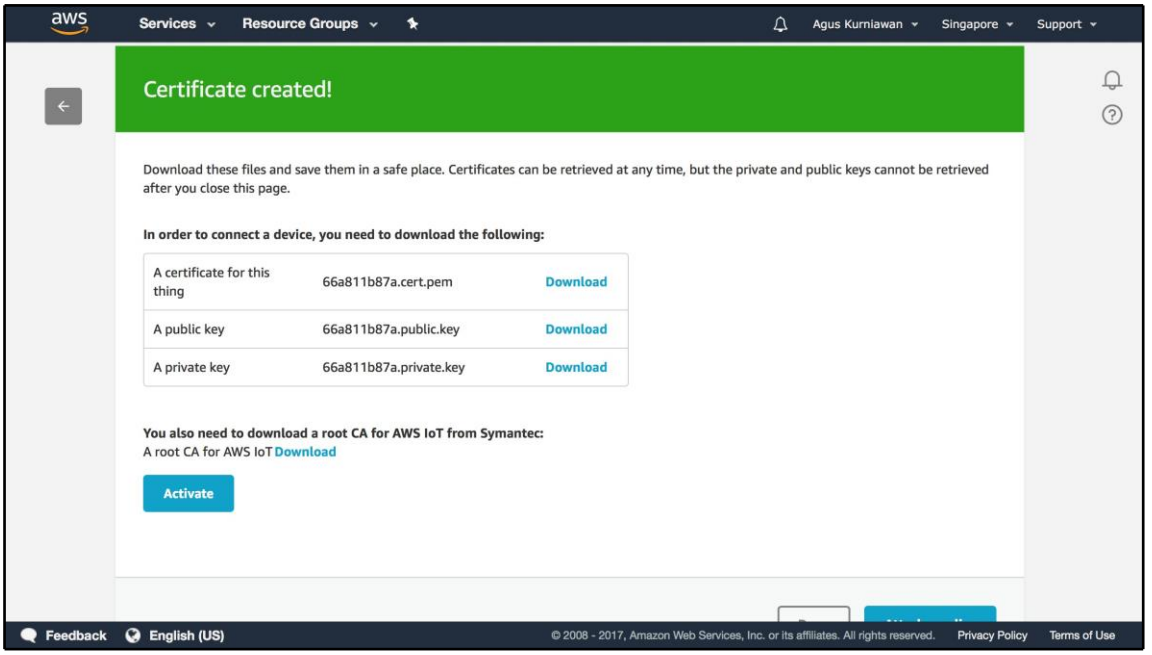
Creating a security certificate

Before we use AWS IoT, we should create a security certificate. Then, this certificate will be attached to our registered IoT device. Follow these steps:

1. On AWS IoT Management Console, open your IoT device. Click on the **Security** option on the left-hand menu. You should see a form, as shown in the following screenshot:



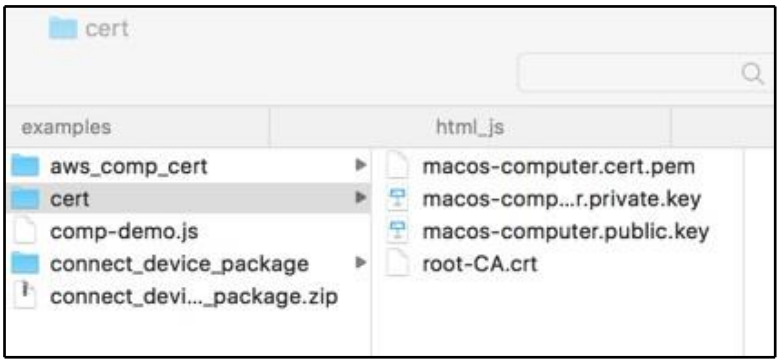
2. You should see a **Create certificate** button. Click on this button. Then, AWS IoT will generate private and public keys for your IoT device. Please download all certificate and key files:



These certificate and key files will be used in our program to access the AWS IoT server. You should get four files, as follows:

- Certificate file (`*.pem`)
- Certificate public key file (`*.key`)
- Certificate private key file (`*.key`)
- Root certificate (`*.pem`) or (`*.crt`)

3. Put all these files into a folder. Our program will access these files:



The next step is to write a program. We will do so in the next section.

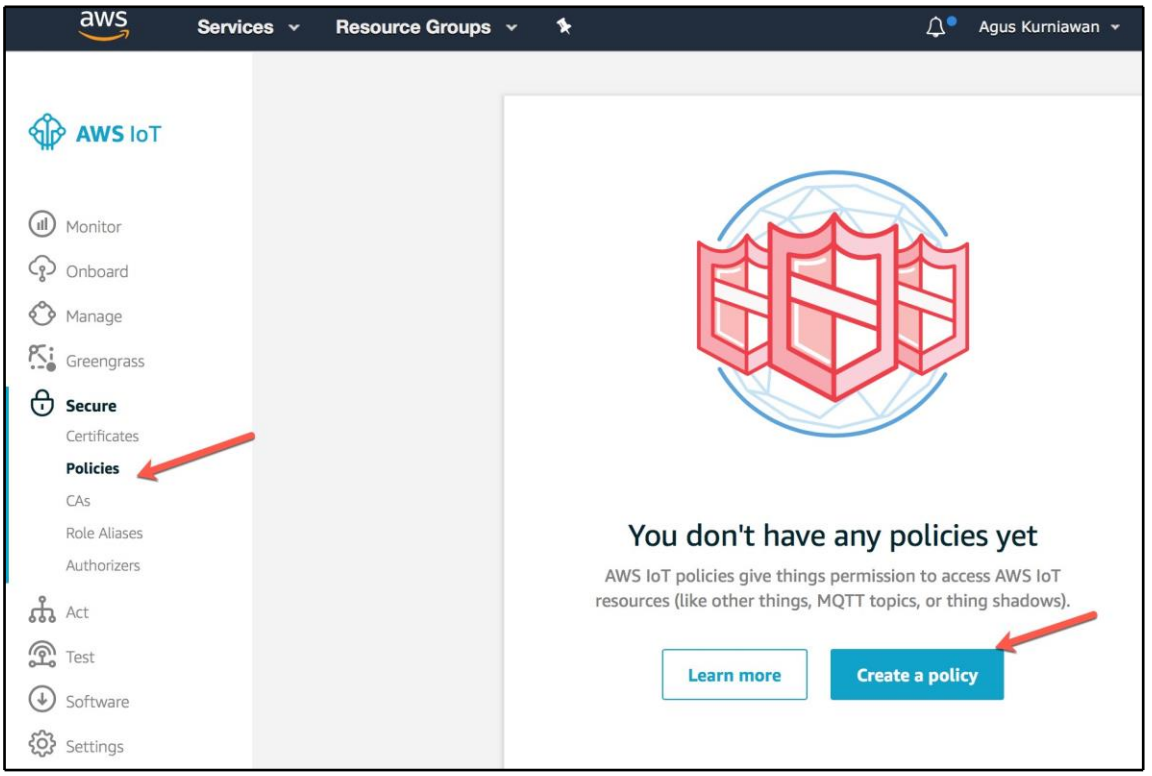
Configuring security access for AWS IoT

Since AWS IoT applies security to protect its system, we should also comply to configure our AWS IoT security. Some steps are taken to configure our AWS IoT security. We will perform the following tasks:

1. Create a policy
2. Attach a policy to the IoT device certificate
3. Attach the IoT thing to the certificate

To create a policy on AWS IoT, perform the following steps:

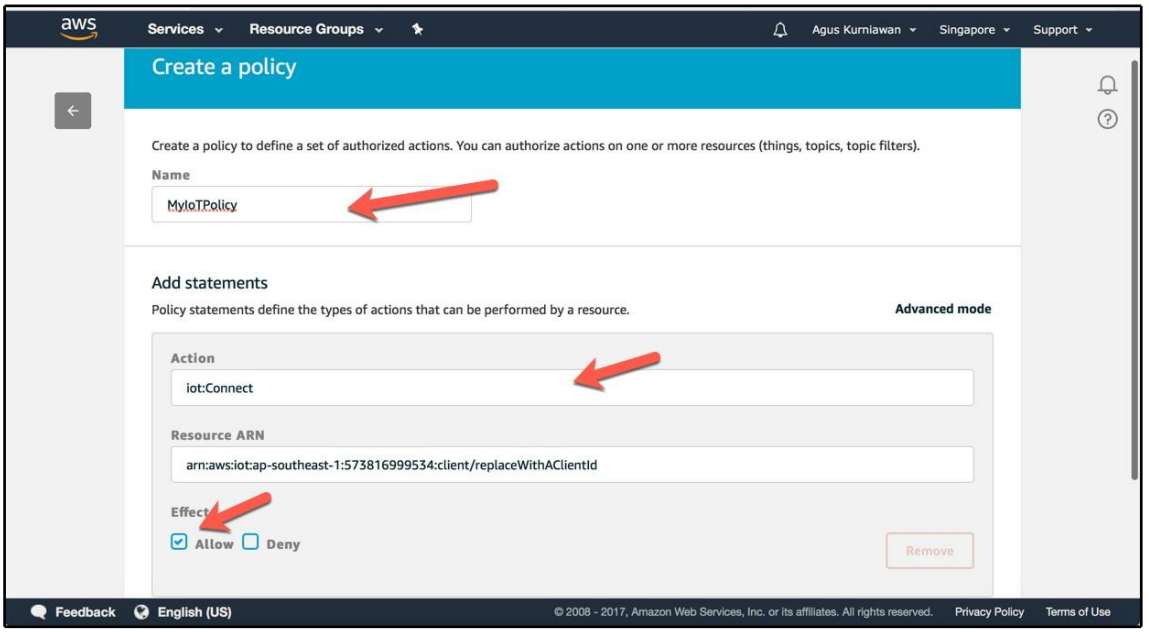
1. Click the **Policies** sub-menu from the **Secure** menu, as shown in the following screenshot:



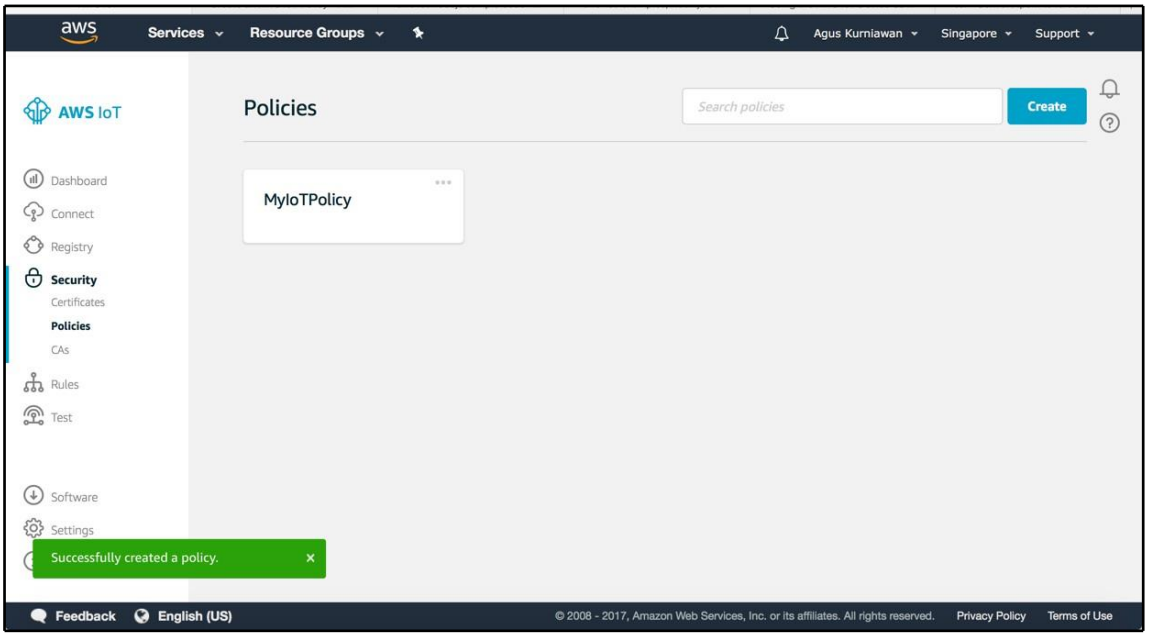
2. Then, you should see a **Create a policy** button. Click on this button.
3. Fill in your policy name. You should add three policy statements, as follows:

    - **iot:Connect**
    - **iot:Subscribe**
    - **iot:Publish**

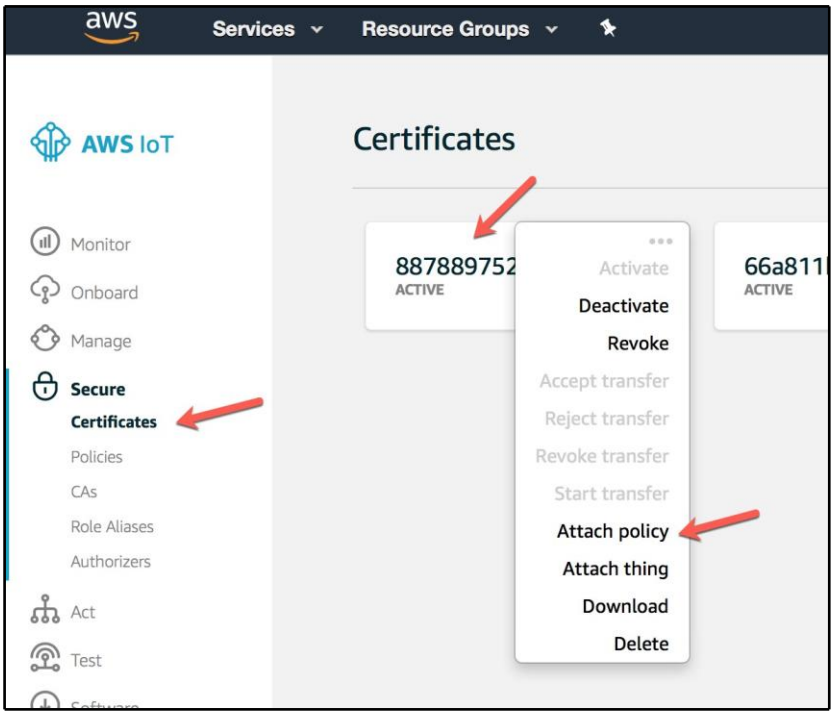
4. Don't forget to check the **Allow** checkbox for all the preceding policy statements:



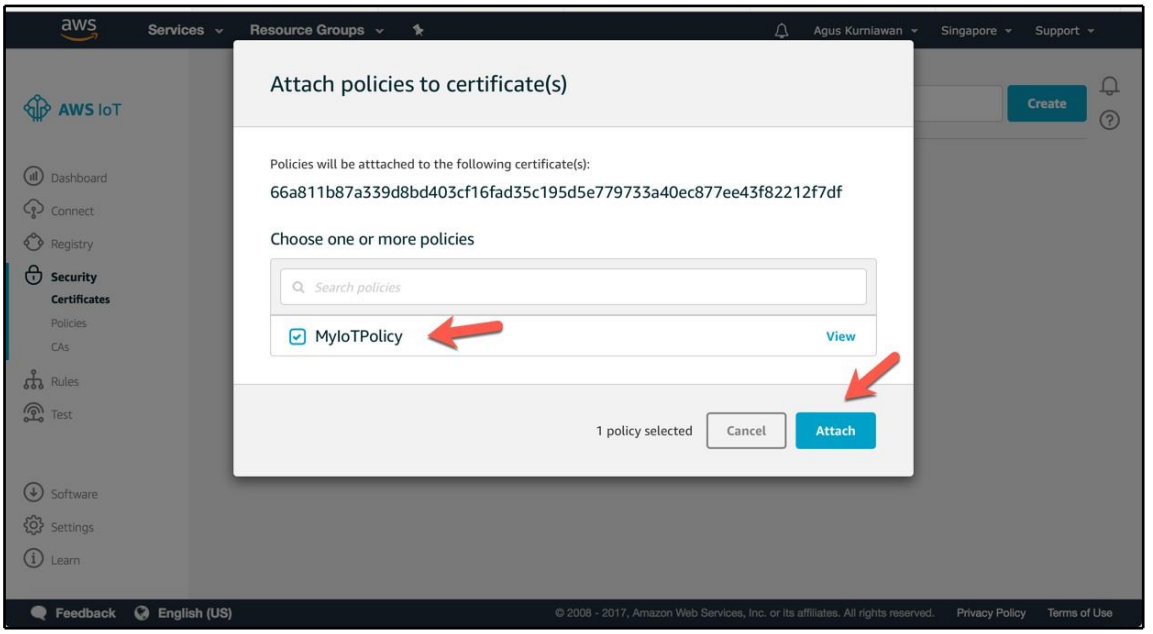
5. When done, save your AWS IoT policy. You should see your created policy on the **Policies** form, as shown in the following screenshot:



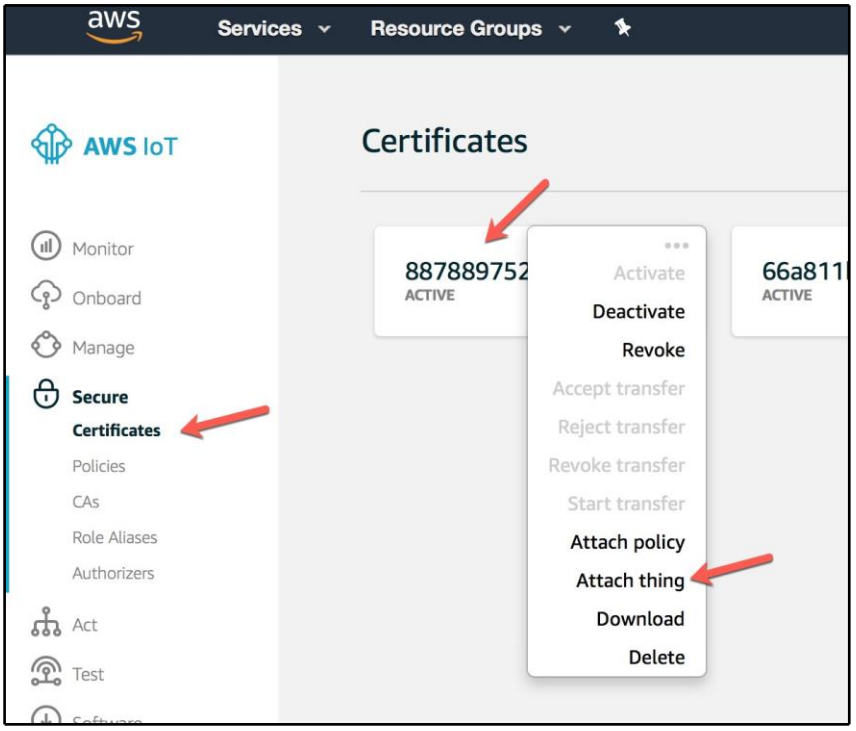
6. The next step is to add our created policy into the IoT device certificate. You can open **Secure** | **Certificates** on AWS IoT Management Console. Click on the ellipsis (…) link so you get a context menu that is shown in the following screenshot. Click on the **Attach policy** option:



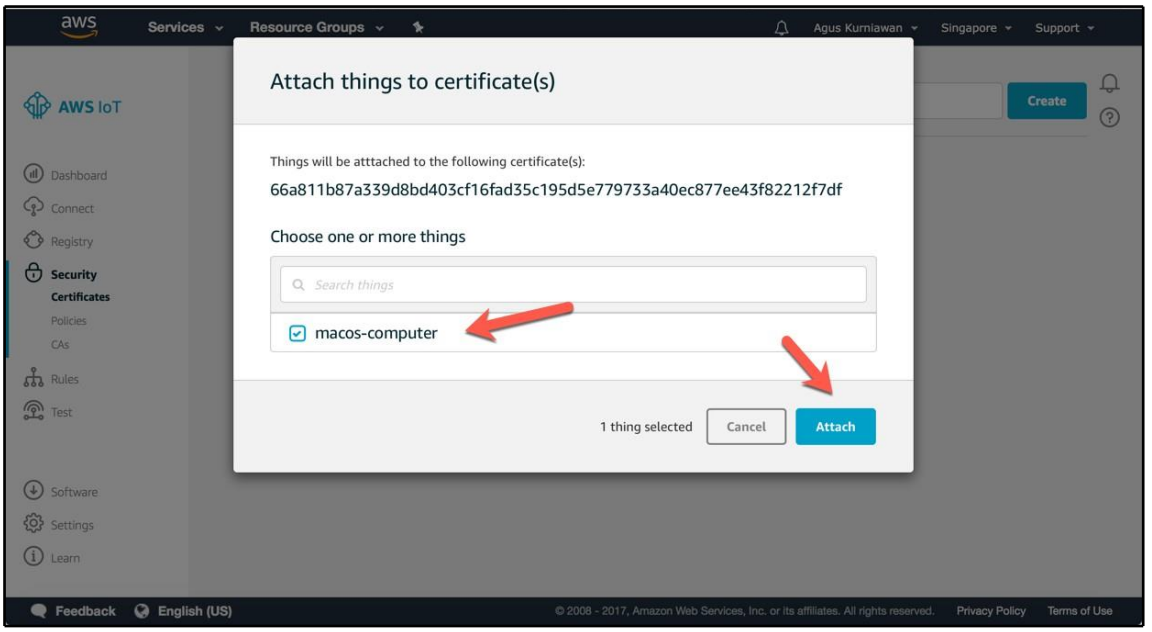
7. Then, you get a dialog box. Select your created policy. When done, click on the **Attach** button to execute this task:



8. The last step is to add our IoT device into a security certificate. Click on the ellipsis (…) on your certificate so you get a context menu. Select the **Attach thing** option on context menu:



9. Select your IoT device and then click on the **Attach** button to perform this task:



Now your IoT device has a certificate and policy. You can access AWS IoT through the IoT device.

Setting up the development environment

After we have registered all the IoT devices for AWS IoT, we can set up our development environment. Depending on your kind of IoT device, you can install AWS IoT SDK for your device. You can review the details at `https://aws.amazon.com/iot/sdk/`.

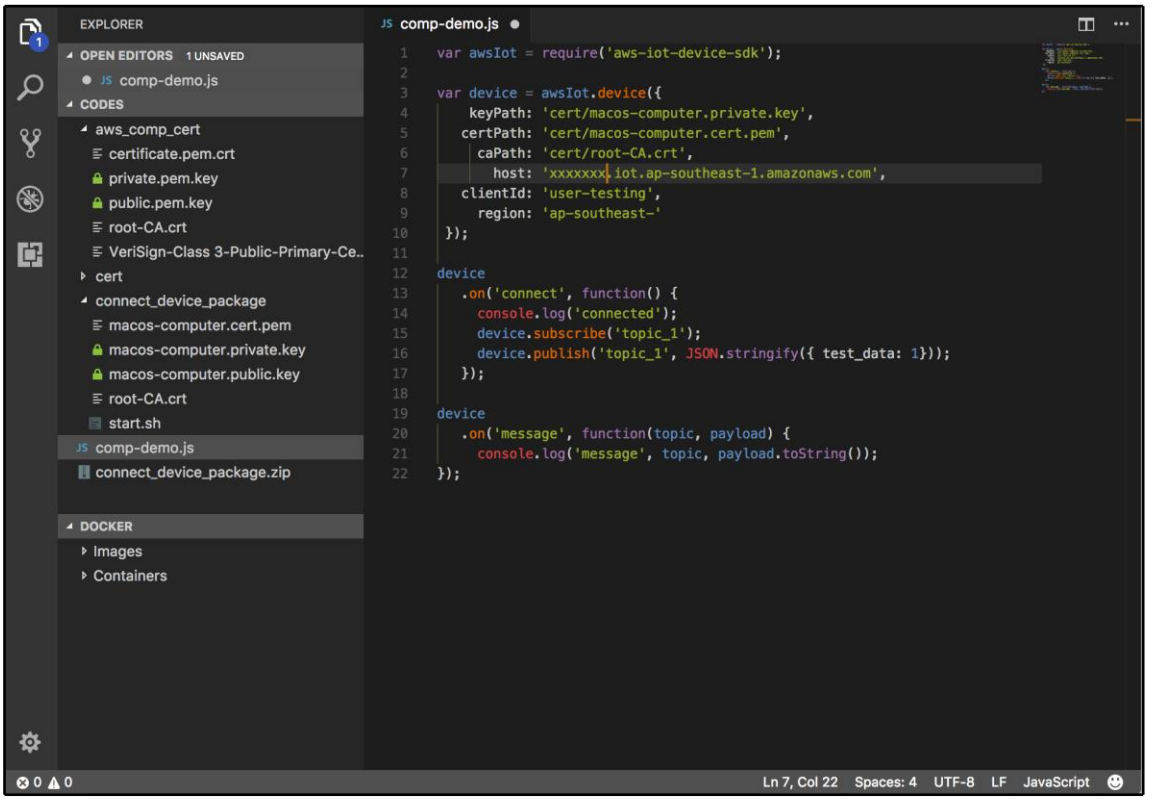
For testing, we use a computer that acts as an IoT thing. The computer will access AWS IoT. To simplify our case, I use JavaScript SDK for AWS IoT, available at `https://github.com/aws/aws-iot-device-sdk-js`. Since JavaScript SDK for AWS IoT needs Node.js to run the program, your computer should install Node.js runtime. You can download and install Node.js for your platform from `https://nodejs.org/`.

To install AWS IoT SDK for JavaScript, do so through the `npm` package. You should install Node.js runtime for your platform. You can type the following command to install AWS IoT SDK for JavaScript:

```
$ npm install aws-iot-device-sdk
```

You can probably run this command at an administrator level if you get an error message due to a security issue.

For the development tool, you can use any text editor to write JavaScript scripts. For instance, you can use Visual Studio Code at `https://code.visualstudio.com`. You can see my sample JavaScript scripts on Visual Studio Code IDE in the following screenshot:



Building an AWS IoT program

After we have configured our AWS IoT and added the IoT device, we can develop a program to access AWS IoT. In this scenario, our computer is used as an IoT thing. We also used Node.js to access AWS IoT, so we need to install AWS IoT SDK for JavaScript. For testing, we will build a Node.js application to access AWS IoT for such purposes as connecting, sending, and receiving.

Now, create a file called `comp-demo.js`. Then, write the following Node.js scripts:

```
var awsIot = require('aws-iot-device-sdk');
var device = awsIot.device({
    keyPath: 'cert/macos-computer.private.key',
   certPath: 'cert/macos-computer.cert.pem',
     caPath: 'cert/root-CA.crt',
       host: 'xxxxxxx.iot.ap-southeast-1.amazonaws.com',
   clientId: 'user-testing',
     region: 'ap-southeast-'
 });
device
   .on('connect', function() {
     console.log('connected');
     device.subscribe('topic_1');
     device.publish('topic_1', JSON.stringify({ test_data: 1}));
   });
device
   .on('message', function(topic, payload) {
     console.log('message', topic, payload.toString());
});
```

Please change the path and certificate files from your AWS IoT on parameters such as `keyPath`, `certPath`, `caPath`, `host`, and `region`. Save this file.

How to work with the program?

Now we will review our program, `comp-demo.js`. The following is a list of steps for the program:

1. Firstly, we apply the required library from AWS IoT SDK for JavaScript. Then, we declare our device based on our IoT thing from AWS IoT:

```
var awsIot = require('aws-iot-device-sdk');
var device = awsIot.device({
    keyPath: 'cert/macos-computer.private.key',
```

```
    certPath: 'cert/macos-computer.cert.pem',
      caPath: 'cert/root-CA.crt',
        host: 'xxxxxxx.iot.ap-southeast-1.amazonaws.com',
    clientId: 'user-testing',
      region: 'ap-southeast-'
  });
```

2. We try to connect to AWS IoT. After we are connected, we subscribe a specific topic, for instance, `topic_1`. Then, we send a message by calling the `publish()` function:

```
device
   .on('connect', function() {
     console.log('connected');
     device.subscribe('topic_1');
     device.publish('topic_1', JSON.stringify({ test_data:
1}));
   });
```

3. To receive an incoming message from AWS IoT, we listen to the message event as follows:

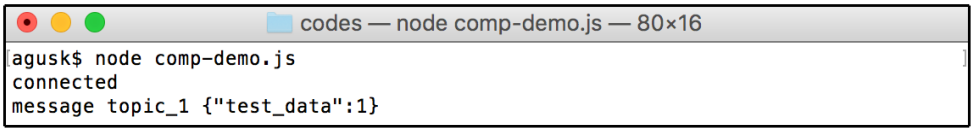
```
device
   .on('message', function(topic, payload) {
     console.log('message', topic, payload.toString());
});
```

Testing all

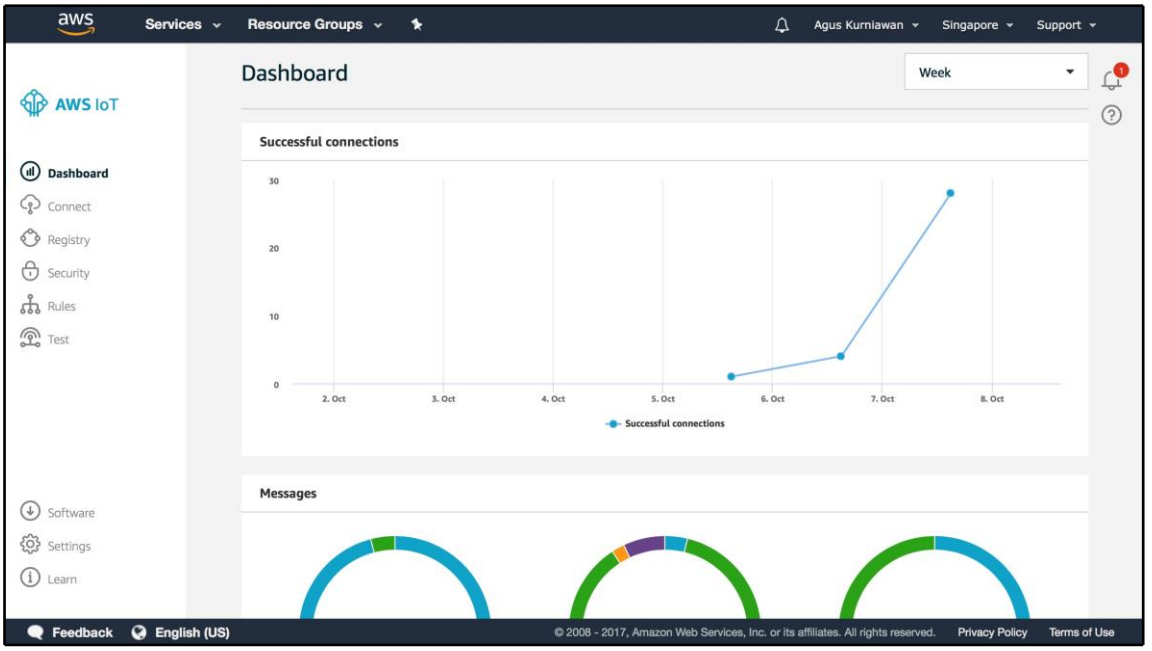
After we write a program, `comp-demo.js`, we can execute this program. Now you can run the program. Type this command:

```
$ node comp-demo.js
```

Make sure all certificate files are on the same path with `comp-demo.js`. If successful, you should see the connected state and receive the incoming message:



You can also verify on AWS IoT Management Console about this transaction:



Summary

We have learned what AWS IoT is and explored several IoT device platforms. AWS IoT Management Console and Device SDK were introduced to obtain the essentials of AWS IoT platform.

In the last section, we registered and configured an IoT device. Then, we created a program to access AWS IoT from the IoT device. In the next chapter, we will learn how to connect to AWS IoT from various IoT devices and make interactions.

Hands On - 2
Connecting IoT Devices to AWS IoT Platform

There are many IoT devices on the market. In this chapter, we will learn how to connect several common IoT devices from the market to AWS IoT. Some tricks and demos are provided to show how to work with these IoT devices. Finally, we will build an IoT application by utilizing IoT devices and AWS IoT.

The following is a list of topics that we will explore in this chapter:

- Introducing a connectivity model for AWS IoT
- Selecting your IoT devices for AWS IoT
- Developing AWS IoT for Raspberry Pi 3
- Developing AWS IoT for Arduino
- Developing AWS IoT for ESP32
- Building an IoT project with AWS IoT

Introducing a connectivity model for AWS IoT

AWS IoT provides several connectivity models to enable IoT devices to establish their connection. Currently, AWS IoT offers the following protocols:

- **Message Queuing Telemetry Transport** (**MQTT**)
- **Hypertext Transfer Protocol** (**HTTP**)
- MQTT over Websocket

We can build a connectivity model between IoT devices and the AWS IoT backend. We can categorize the IoT device platform into the following three models based on their supported connectivity:

- For an IoT device with network capabilities, if this device has all the support required by AWS IoT, then this device can access AWS IoT directly.
- The second connectivity model is to build a connection to AWS IoT through a gateway. This approach is applied for an IoT device that has network capabilities which are not supported for AWS IoT protocols.
- The last approach is to be applied to IoT devices that do not have network capabilities. We should attach these devices to a network device, such as Raspberry Pi, BeagleBone, and a computer that works as a bridge between the device and AWS IoT.

Selecting your IoT devices for AWS IoT

We already know that there are a lot of IoT devices on the market. To work with AWS IoT, Amazon provides guidelines about a list of IoT devices with supported AWS IoT. You can read it at `https://aws.amazon.com/iot-platform/getting-started/#kits`. Otherwise, you can use own IoT devices and ensure that they are compatible with AWS IoT SDK by referring to `https://aws.amazon.com/iot-platform/sdk/`.

From this, you can make a self-assessment when you select an IoT device for the AWS IoT platform. You can verify your selection with the following questions:

- *Does the IoT device have a network capability?* A network capability probably can identity network features such as Ethernet, Wi-Fi, Bluetooth, Zigbee and LoRa.
- *Does the IoT device network module give support for the AWS IoT protocol?* Make sure your IoT device can communicate with external devices through HTTP, MQTT, or MQTT over Websocket.

If your IoT device does not fit these criteria, your IoT device can connect to AWS IoT using a third approach—attaching your IoT device to a network device, for instance, a computer, and then connecting from a program inside the computer to AWS IoT.

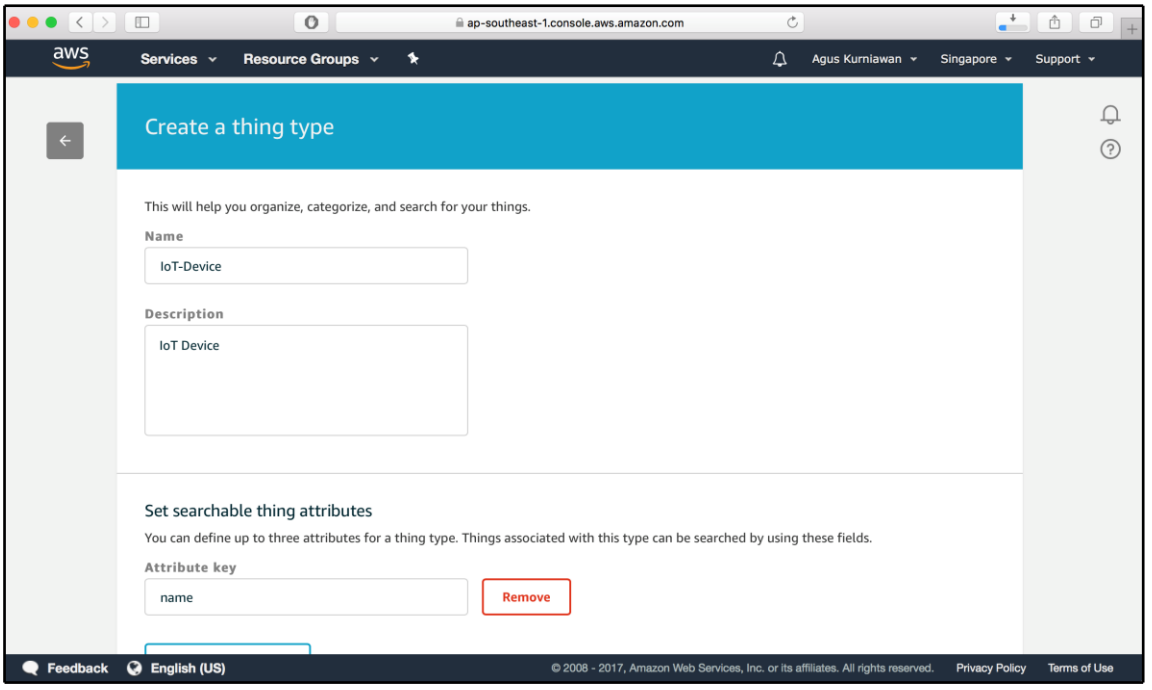
We will explore and review some IoT devices to connect AWS IoT in the next section. The following are the three IoT devices that we will review in this chapter:

- Raspberry Pi 3
- Arduino Yún
- ESP32 board

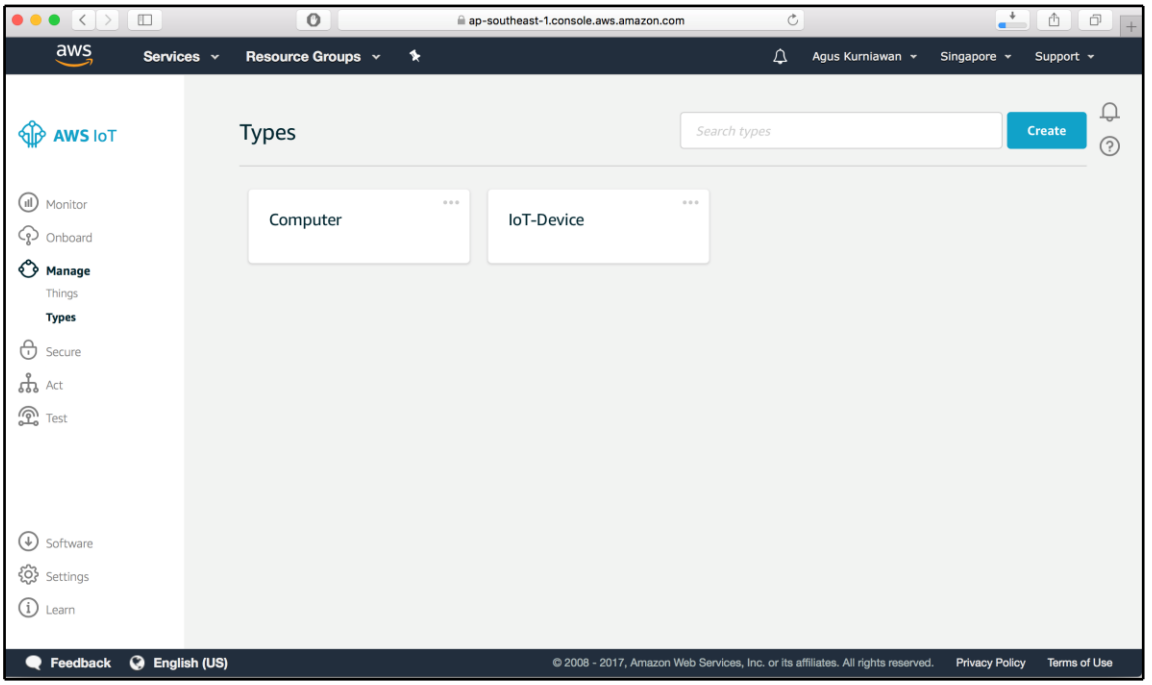
Configuring IoT devices to AWS IoT

In this section, we will configure an IoT device in order to access AWS IoT. Several steps should be taken to complete the registration process:

1. Firstly, we create a thing type. You can perform this task on the **Manage** | **Types** menu from AWS IoT Console and then click on the **Create** button. Fill in the type name and attributes for your IoT device. For instance, we give a thing type as `IoT-Device` with `name` as the thing attribute :

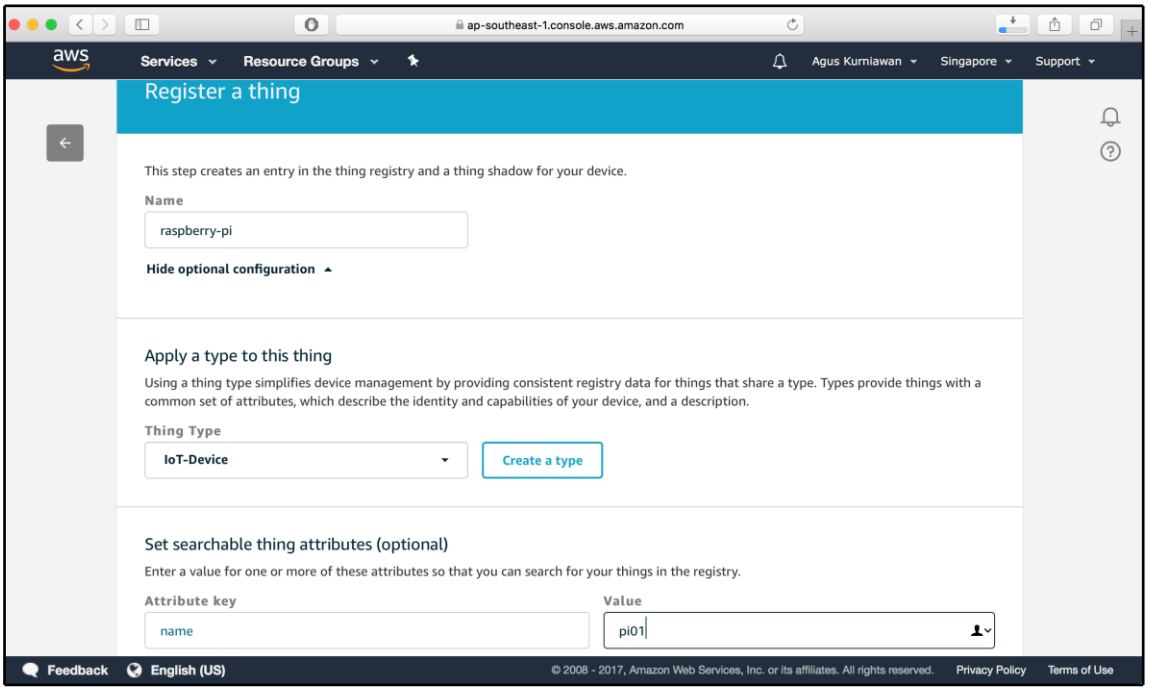


2. After this has been created, you should see it on the **Types** screen from the AWS IoT dashboard, as shown in the following screenshot:

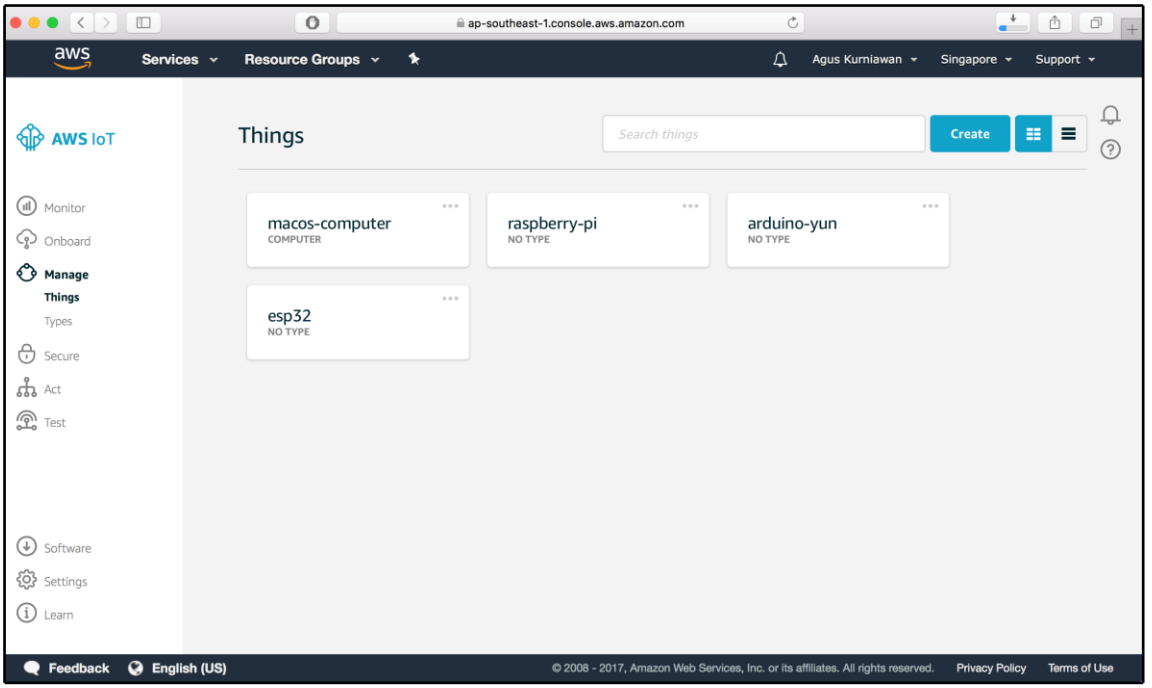


3. Next, we add three IoT devices into AWS IoT. Navigate to **Manage** | **Things** on the AWS IoT dashboard. Then, you add a thing name and its attribute as shown in the screenshot below the table. The following is our configuration of IoT devices:

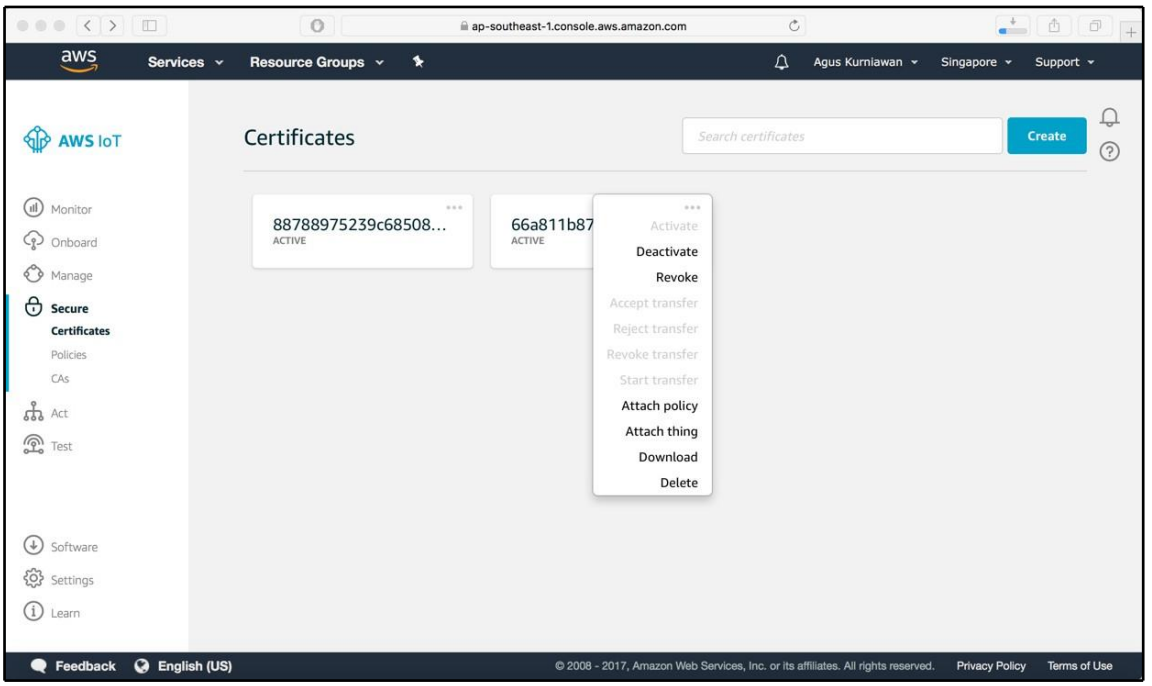
Device name	**Thing name**	**Thing attribute**
Raspberry Pi 3	`raspberry-pi`	`pi01`
Arduino Yún	`arduino-yun`	`arduino01`
ESP32 board	`esp32`	`esp32-01`



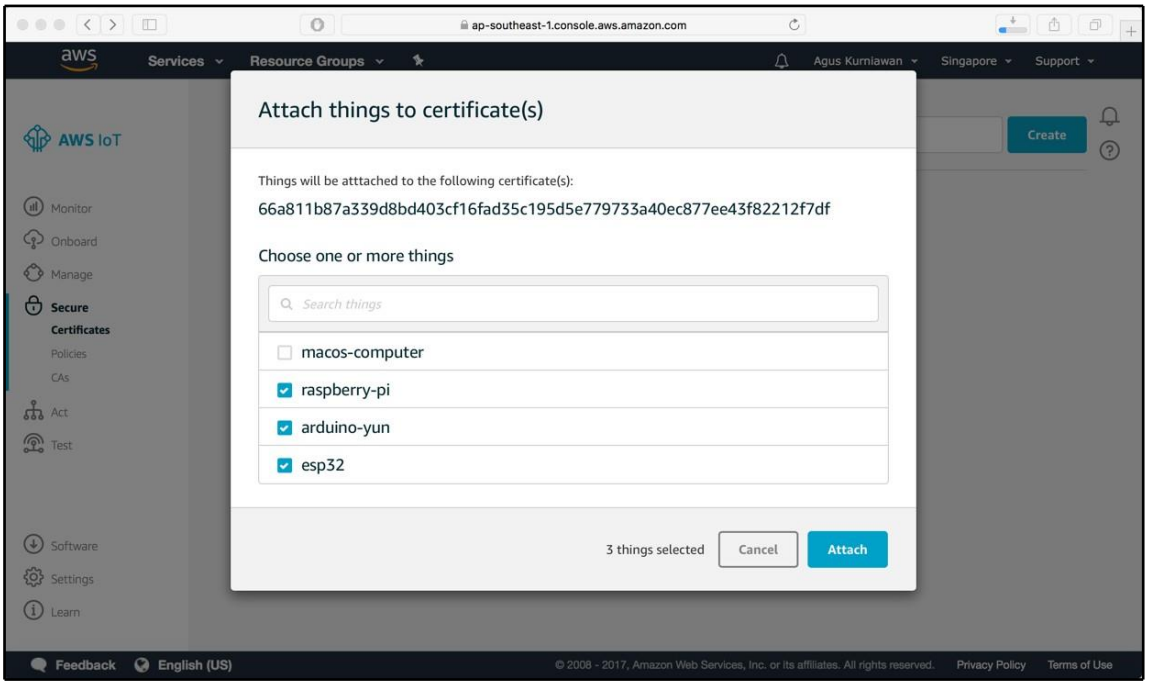
4. After we have added three IoT devices, you should see these devices on the **Things** dashboard from the AWS IoT Management Console, as shown in the following screenshot:



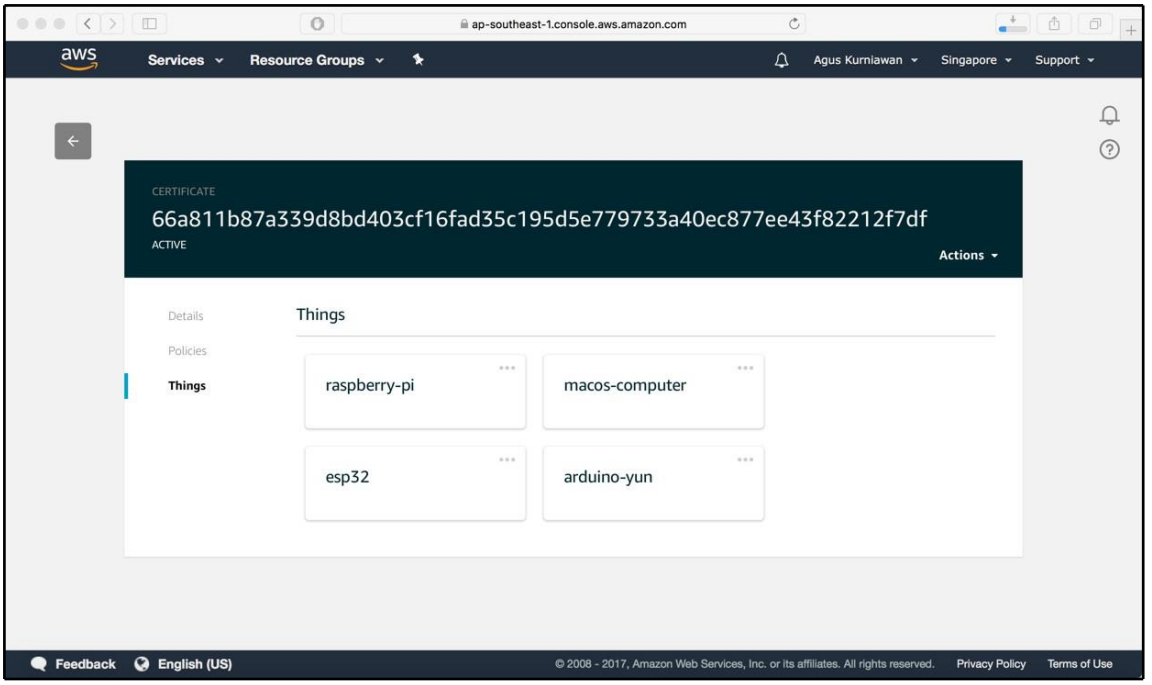
5. The next step is to add our IoT devices to a security certificate. Since we have created a security certificate in `Chapter 1`, *Getting Started with AWS IoT*, we can use that. We add our three devices into our existing certificate by clicking on the certificate, which locates on the **Secure** | **Certificates** menu from AWS IoT, so you should see a context menu, shown in the following screenshot:



6. Select the **Attach thing** menu on the context menu. Then, check all the devices. Click on the **Attach** button once done, as shown in the following screenshot:



7. We can verify whether our devices are already added or not. You can check it on the **Things** screen from your certificate on AWS IoT Management Console, as shown in the following screenshot:



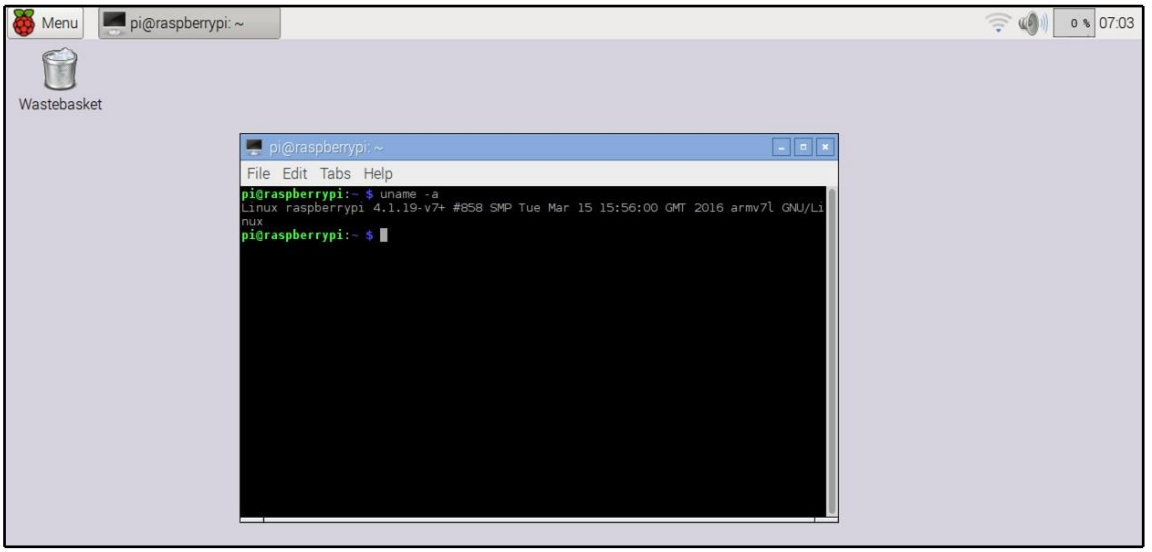
Now our devices are ready for AWS IoT development.

AWS IoT development for Raspberry Pi 3

Raspberry Pi is a famous IoT development. At a low price, you can obtain a minicomputer. Some demo scenarios in this book can be implemented using a desktop computer. Currently, Raspberry Pi 3 is the latest product from Raspberry Pi Foundation. For further information about Raspberry Pi and getting this board, I recommend that you visit the official website at `http://raspberrypi.org`.

Raspberry Pi 3 can be deployed on various OSes, such as Linux and Windows 10 IoT Core. In this chapter, we will focus on Raspberry Pi 3 with Linux as the OS. We use Raspbian Linux. You can download and install it at `https://www.raspberrypi.org/downloads/raspbian/`.

Raspbian Linux can work as a console or in desktop mode. You can see Raspbian Linux on Raspberry Pi 3 in the following screenshot:



The corresponding AWS IoT SDK is available at `https://aws.amazon.com/iot-platform/sdk/`. There are SDKs for Raspberry Pi 3 with the following approaches:

- **AWS IoT SDK JavaScript**: This is used if you want to develop a program-based JavaScript, such as Node.js
- **AWS IoT SDK Python**: If you want to develop Python for AWS IoT, you can use this SDK on your Raspberry Pi 3
- **AWS IoT SDK Java**: For Java developers, this SDK can extend your skill to develop Java applications to connect to AWS IoT
- **AWS IoT SDK Embedded C**: If you are familiar with C programming, you can choose this SDK to develop an AWS IoT program

Each AWS IoT SDK has a different approach to deploy it on your device. You can follow the guidelines in the documentation for each AWS IoT SDK.

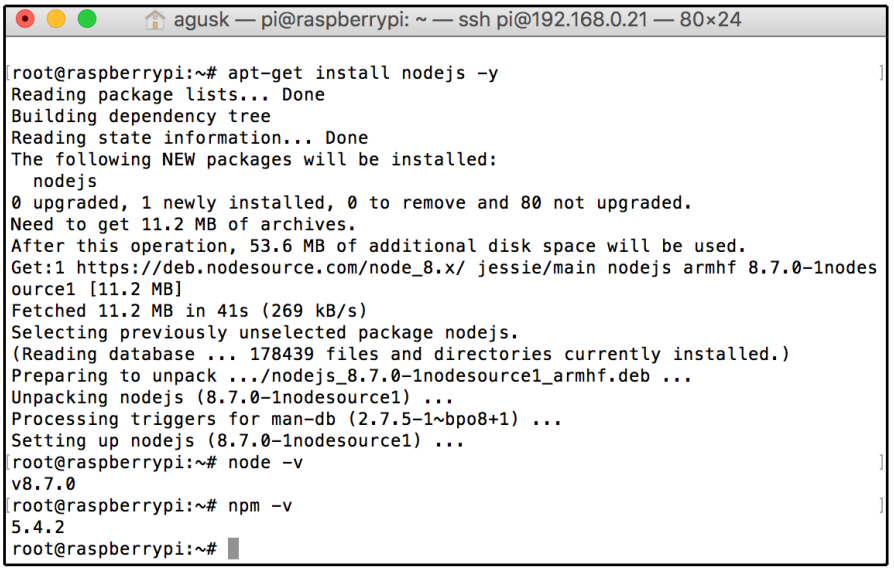
In this section, we use AWS IoT SDK JavaScript on Raspberry Pi 3. The AWS IoT SDK JavaScript installation process is the same as the installation process on a computer. Make sure your Raspberry Pi has Node.js installed. Since Raspbian Linux on Raspberry Pi has Node.js legacy installed, we should uninstall it and then install the latest Node.js. You can uninstall Node.js legacy by typing the following commands:

```
$ sudo apt-get remove nodered -y
$ sudo apt-get remove nodejs nodejs-legacy -y
$ sudo apt-get remove npm -y
```

To install the latest Node.js on Raspberry Pi, we can use Node.js from the Nodesource distributor, available at `https://github.com/nodesource/distributions`. Type the following commands:

```
$ sudo su -
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo bash -
$ apt-get install nodejs -y
$ node -v
$ npm -v
$ exit
```

If successful, you should see the latest Node.js and **Node Package Manager** (**NPM**), as shown in the following screenshot:



After you have the latest Node.js on Raspberry Pi 3, you can install AWS IoT SDK JavaScript through NPM. This SDK can be found at `https://github.com/aws/aws-iot-device-sdk-js`. To install this SDK, type the following command on the Terminal of Raspberry Pi 3:

```
$ npm install aws-iot-device-sdk
```

For testing, we use the same program from `Chapter 1`, *Getting Started with AWS IoT*. You can write this complete program. You should change paths for the certificate, private key, and AWS IoT host. These security certificate files are obtained from AWS IoT. You can read about this in `Chapter 1`, *Getting Started with AWS IoT*.

The following is the complete program for our testing:

```
var awsIot = require('aws-iot-device-sdk');

var device = awsIot.device({
 keyPath: 'cert/private.key',
 certPath: 'cert/cert.pem',
 caPath: 'cert/root-CA.pem',
 host: '<host-aws-iot-server>',
 clientId: 'user-testing',
 region: 'ap-southeast-'
 });

device
 .on('connect', function() {
 console.log('connected');
 device.subscribe('topic_1');
 device.publish('topic_1', JSON.stringify({ test_data: 1}));
 });

device
 .on('message', function(topic, payload) {
 console.log('message', topic, payload.toString());
});
```

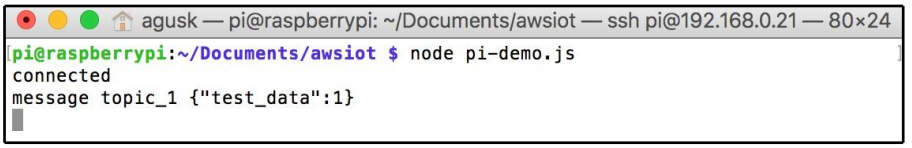
Save this program as the `pi-demo.js` file.

The screenshot of the preceding code is shown as follows:

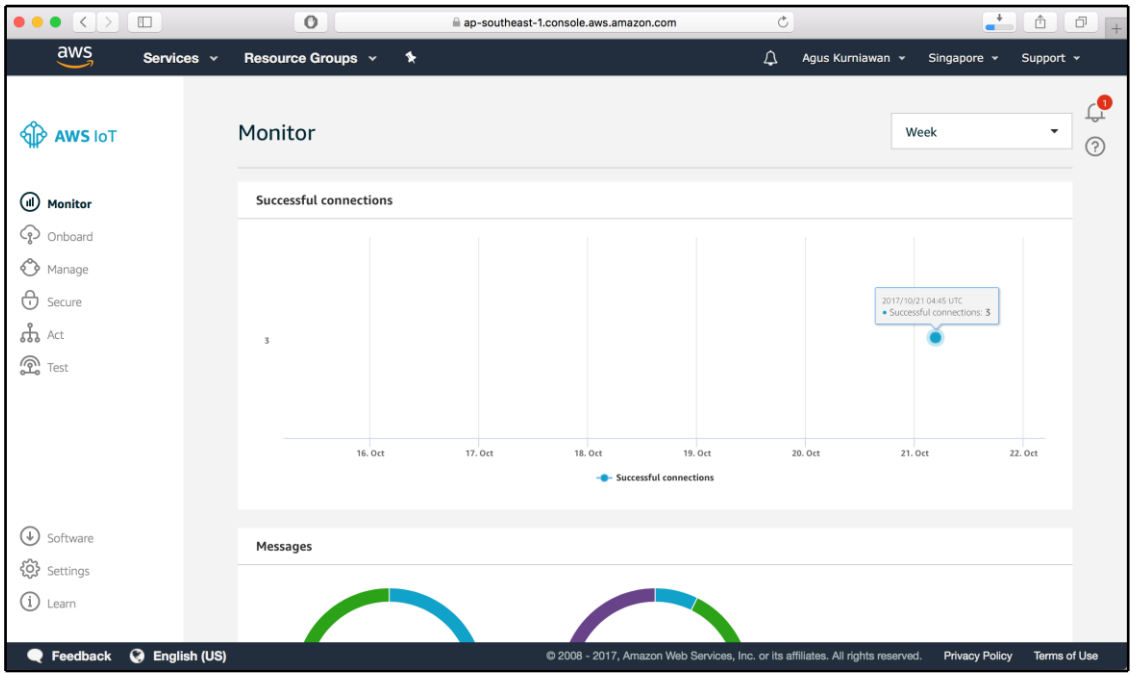
Now you can run this program by typing the following command:

```
$ node pi-demo.js
```

If we succeed, the program will present a message, as shown in the following screenshot:



You also can verify this by opening the **Monitor** dashboard from AWS IoT Management Console. You should see a number of connections to your server:



AWS IoT development for Arduino

Most IoT device developments use the Arduino board as the board target. Since Arduino shares schemes and designs, the makers make their own boards with custom Arduino models. Arduino models range from the basic Arduino board model to the advanced Arduino board model. The Arduino board is the best choice to introduce IoT development.

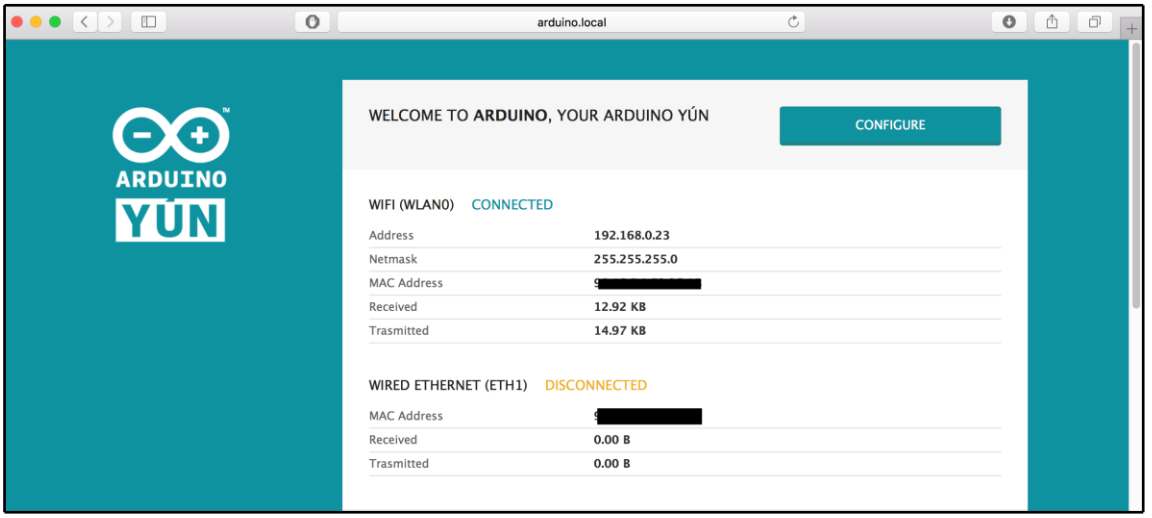
In this section, we will explore how to use Arduino Yún to interact with AWS IoT. For further information about Arduino Yún, I recommend that you read the official documentation at `https://www.arduino.cc/en/Guide/ArduinoYun`.

To work with AWS IoT using Arduino Yún, we perform the following steps:

1. Install the Arduino software
2. Connect Arduino Yún to a network through Ethernet or a Wi-Fi module
3. Configure Arduino Yún for AWS IoT SDK
4. Build a program for Arduino Yún

Each step will be performed. Now, you can follow the steps in detail, as follows:

1. Firstly, you should install the Arduino software for your OS platform. You can download it at `https://www.arduino.cc/en/Main/Software`.
2. The next step is to connect your Arduino Yún to the existing network. You should join your Arduino Yún into an existing Wi-Fi hotspot. Then, configure your Arduino Yún to join the Wi-Fi network. You can also connect this board to a network through the Ethernet module:



Make sure your computer and Arduino Yún have the same network segment so you can access Arduino Yún over the network.

3. The next step is to configure Arduino Yún for AWS IoT. You should install AWS IoT SDK for Arduino Yún at `https://github.com/aws/aws-iot-device-sdk-arduino-yun`. You can download the latest AWS IoT SDK for Arduino Yún at `https://s3.amazonaws.com/aws-iot-device-sdk-arduino-yun/AWS-IoT-Arduino-Yun-SDK-latest.zip`.
4. Extract a ZIP file into a certain folder. In `Chapter 1`, *Getting Started with AWS IoT*, we have the certificate and private key files for AWS IoT. Put them in the `AWS-IoT-Arduino-Yun-SDK/AWS-IoT-Python-Runtime/certs` folder from the extracted AWS IoT SDK for Arduino Yún .
5. Now, we configure and upload our certificate and private key files into Arduino Yún. For Mac and Linux, you can open the Terminal and navigate to a folder where the SDK file is extracted. You should see the `AWSIoTArduinoYunInstallAll.sh` file. Then, execute this file using the following commands:

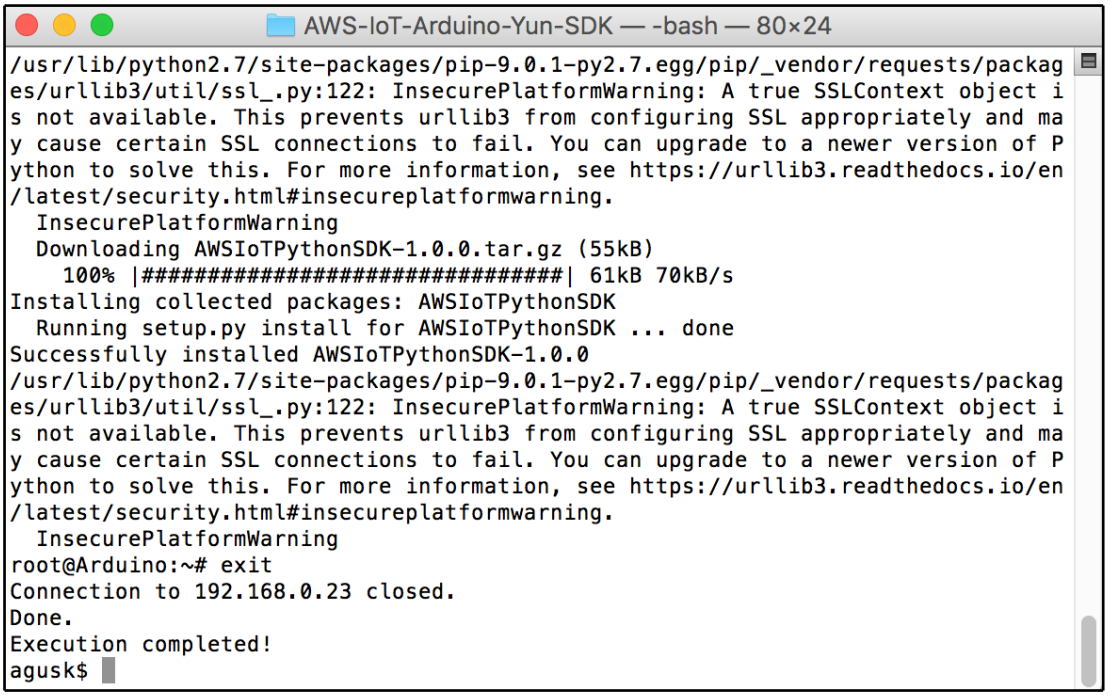
```
$ chmod 755 AWSIoTArduinoYunInstallAll.sh
$ ./AWSIoTArduinoYunInstallAll.sh <Board IP> <UserName> <Board
Password>
```

   In the preceding code, `<Board IP>` is the IP address of your Arduino Yún. `<UserName>` and `<Board Password>` are the username and password to access your Arduino Yún. By default, if you do not change the password, the username is `root` and the password is `arduino`. This process will take several minutes. Please do not close the process.

6. For Windows, you can use Putty and WinSCP to configure AWS IoT on Arduino Yún . You can download Putty at `http://www.chiark.greenend.org.uk/%7Esgtatham/putty/download.html` and WinSCP at `http://winscp.net/eng/download.php`. Then, you can type the following commands:

```
$ opkg update
$ opkg install distribute
$ opkg install python-openssl
$ easy_install pip
$ pip install AWSIoTPythonSDK==1.0.0
```

The following is a sample of the executing output on macOS:



7. For a demo, we use a sample program from AWS IoT SDK for Arduino Yún. It is called `BasicPubSub`. You can find it on the **File| Examples| AWS-IoT-Arduino-Yun-Library** menu from the Arduino IDE. You should see the `BasicPubSub` menu. Click on it so you can open the `BasicPubSub` program, as shown in the following screenshot:

8. You can see that there are two files—`BasicPubSub.ino` and `aws_iot_config.h`. The following is a program from the `BasicPubSub.ino` file:

```
...

aws_iot_mqtt_client myClient; // init iot_mqtt_client
char msg[32]; // read-write buffer
int cnt = 0; // loop counts
int rc = -100; // return value placeholder
bool success_connect = false; // whether it is connected

// Basic callback function that prints out the message
void msg_callback(char* src, unsigned int len, Message_status_t
flag) {
 if(flag == STATUS_NORMAL) {
 Serial.println("CALLBACK:");
 int i;
 for(i = 0; i < (int)(len); i++) {
 Serial.print(src[i]);
 }
 Serial.println("");
 }
}

...
void loop() {
 if(success_connect) {
 // Generate a new message in each loop and publish to "topic1"
 sprintf(msg, "new message %d", cnt);
 if((rc = myClient.publish("topic1", msg, strlen(msg), 1,
false)) != 0) {
 Serial.println(F("Publish failed!"));
 Serial.println(rc);
 }

...

 delay(1000);
```

9. Now we can open the `aws_iot_config.h` file. We should configure the certificate and private key files on the `aws_iot_config.h` file. Change `AWS_IOT_MQTT_HOST` for the IoT endpoint. Also, change the `AWS_IOT_CERTIFICATE_FILENAME` and `AWS_IOT_PRIVATE_KEY_FILENAME` values for the certificate and private key files for your IoT device:

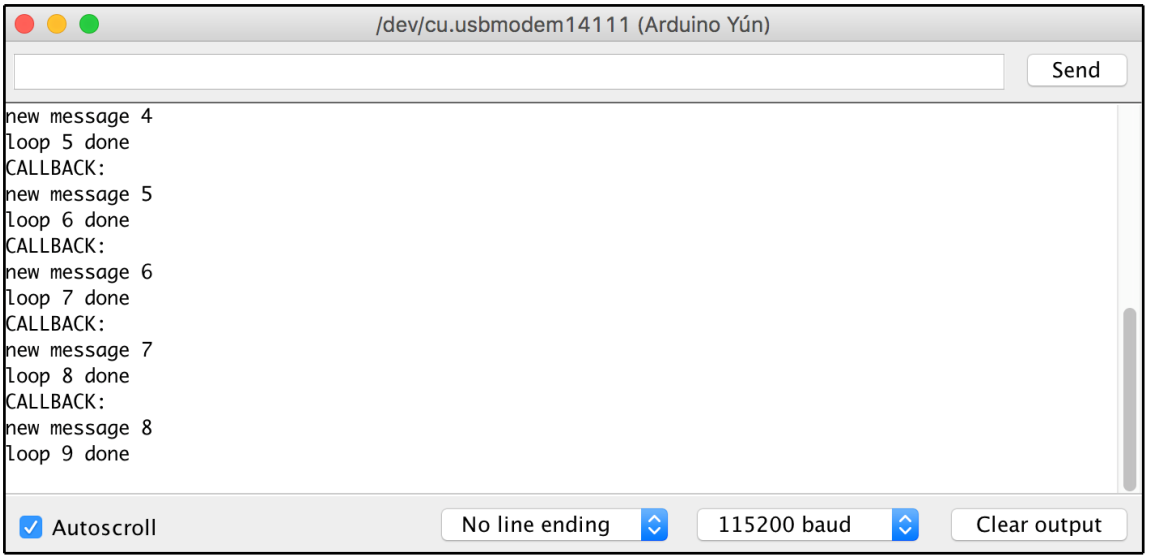
```
#ifndef config_usr_h
#define config_usr_h

// Copy and paste your configuration into this file
//===========================================================
==
#define AWS_IOT_MQTT_HOST “<aws-iot-host>” // your endpoint
#define AWS_IOT_MQTT_PORT 8883 // your port
#define AWS_IOT_CLIENT_ID "My_ClientID" // your client ID
#define AWS_IOT_MY_THING_NAME "My_Board" // your thing name
#define AWS_IOT_ROOT_CA_FILENAME "root-CA.crt" // your root-CA
filename
#define AWS_IOT_CERTIFICATE_FILENAME “thing.cert.pem" // your
certificate filename
#define AWS_IOT_PRIVATE_KEY_FILENAME “private.key" // your
private key filename
//===========================================================
==
// SDK config, DO NOT modify it
#define AWS_IOT_PATH_PREFIX "../certs/"
#define AWS_IOT_ROOT_CA_PATH AWS_IOT_PATH_PREFIX
AWS_IOT_ROOT_CA_FILENAME // use this in config call
#define AWS_IOT_CERTIFICATE_PATH AWS_IOT_PATH_PREFIX
AWS_IOT_CERTIFICATE_FILENAME // use this in config call
#define AWS_IOT_PRIVATE_KEY_PATH AWS_IOT_PATH_PREFIX
AWS_IOT_PRIVATE_KEY_FILENAME // use this in config call

#endif
```

Save all the changes.

10. To execute the program, compile and upload this sketch program into Arduino Yún. After it is uploaded, you can open the Serial Monitor tool with baud rate 115200. You can see the program output on the Serial Monitor tool:



You can verify that this is executing by opening AWS IoT Console to see the incoming messages from Arduino Yún .

AWS IoT development for boards based on ESP32

The ESP32 chip is an MCU with Wi-Fi and Bluetooth network modules. This chip is developed by Espressif. For further information about the ESP32 chip, I recommend that you read the product documentation at `http://espressif.com/en/products/hardware/esp32/overview`.

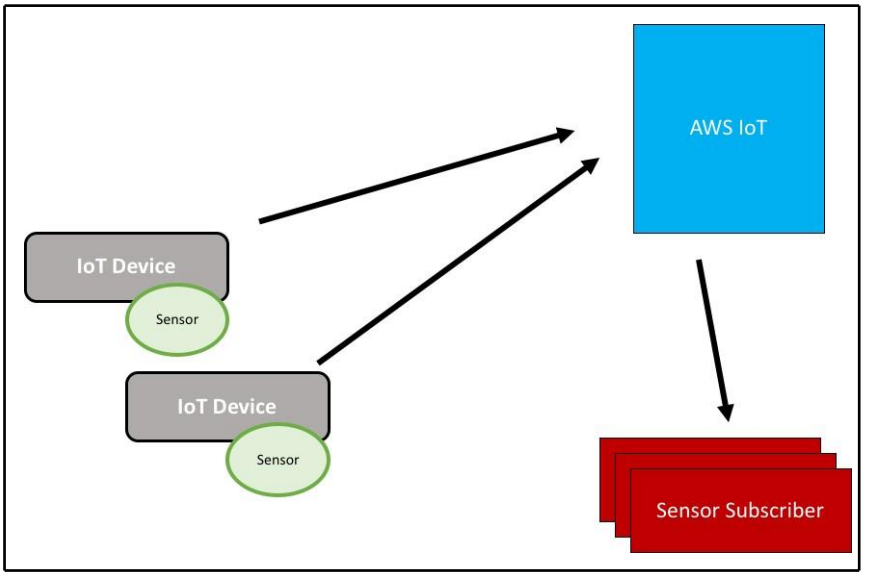
There are a lot of boards based on ESP32. You can find various boards based on ESP32 from SparkFun, Adafruit, or embedded companies from China. To work with AWS IoT on the ESP32 chip, we can apply the Mongoose OS platform from Cesanta. You can review the details of this platform on the official website at `https://mongoose-os.com`.

Cesanta provides an IoT kit-based ESP32 chip, called **Mongoose OS ESP32-DevKitC**. The software of Mongoose OS is built from Node.js/JavaScript. To configure this board connecting to AWS IoT, you can follow the instructions at `https://mongoose-os.com/software.html`.

Building an IoT project with AWS IoT

In this section, we will try to develop a simple IoT application by applying the AWS IoT platform. We will deploy the IoT program on several IoT devices. These devices will use sensors to sense the physical environment, and then send a result of the sensing to AWS IoT. We will also create a small application to listen to incoming messages from AWS IoT. This is called the **sensor subscriber** application.

You can see a general demo architecture in the following figure. For testing, we use Arduino Yún as the IoT device node:



We will build each component in the next section.

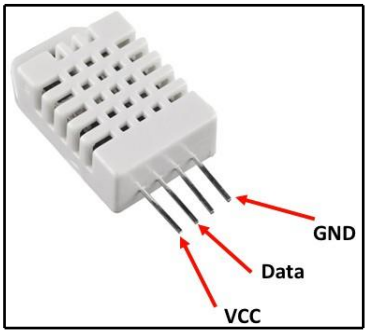
Configuring AWS IoT

We should configure your AWS IoT. We already configured this in `Chapter 1`, *Getting Started with AWS IoT*, and in the first section in this chapter. The idea is to register each IoT device and application to AWS IoT Console. After configuration, we should get the certificate and private key files that will apply in our program.

Developing the Arduino program

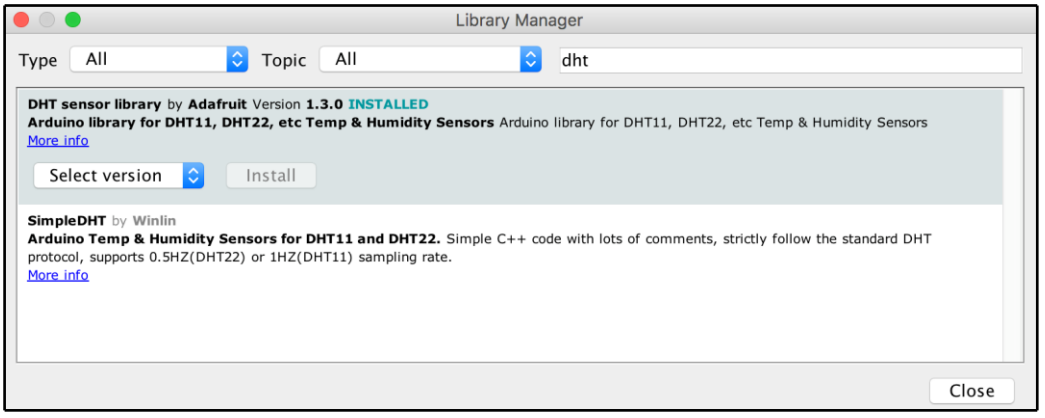
For the IoT node device, we apply Arduino Yún with the DHT22 sensor that provides temperature and humidity sensing. You can find DHT22 easily on an online electronic store or at a local store.

The general DHT22 sensor scheme is described in the following image. The third pin is unused. This sensor will be attached to Arduino Yún :



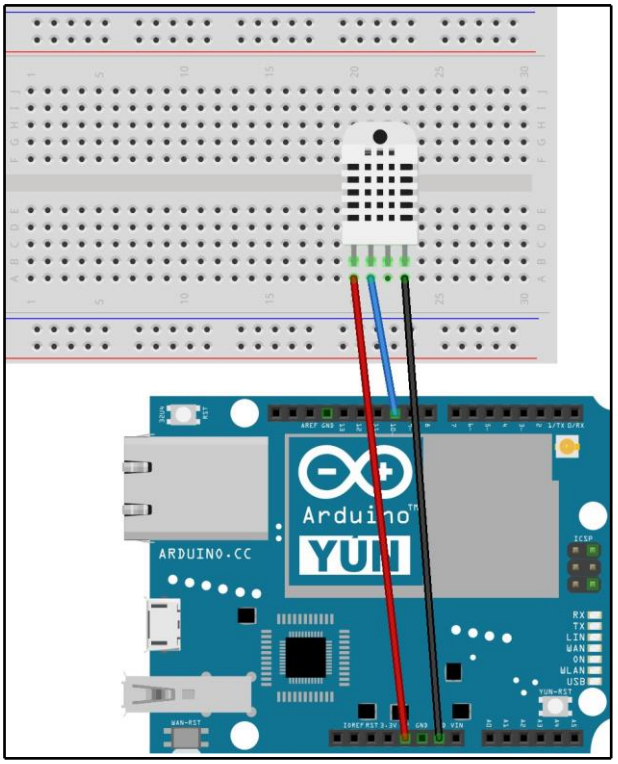
Now we can write the Arduino program. You can proceed with the following steps:

1. To use the DHT sensor with the Arduino board, we use the DHT library from Adafruit. You can find it at`https://github.com/adafruit/DHT-sensor-library`. You can install it by downloading the DHT source from GitHub. You also can install it through the Arduino IDE library. Type `DHT` and select DHT library from Adafruit:



2. To work with the DHT library from Adafruit, you should download the `Adafruit_Sensor.h` file from `https://github.com/adafruit/Adafruit_Sensor` . Copy `Adafruit_Sensor.h` into the `DHT_sensor_library` folder in the Arduino libraries.
3. Now we build the wiring for our demo. Attach DHT22 into a breadboard, and then connect to Arduino through jumper cables. The following is our wiring:
    - DHT22 VCC is connected to Arduino 5V pin
    - DHT22 GND is connected to Arduino GND pin
    - DHT22 Data is connected to Arduino digital pin 10

You can see the wiring implementation in the following image:



4. The next step is to develop a sketch program. We modify from the previous program, `BasicPubSub`. We change the `topic1` channel to the `sensorroom` channel. Before we send data to AWS IoT, we read temperature through DHT22:
    1. Firstly, we add the DHT library and initialize our library:

```
#include "DHT.h"
#define DHTTYPE DHT22
#define DHTPIN 10
DHT dht(DHTPIN, DHTTYPE);

void setup() {
 ....
 dht.begin();

....
 if((rc = myClient.subscribe("sensorroom", 1,
msg_callback)) != 0) {
 Serial.println("Subscribe failed!");
 Serial.println(rc);
 }
}
```

    2. On the `loop()` function, we read the temperature and then send it to AWS IoT:

```
void loop() {
 if(success_connect) {

 char floatBuffer[8];
 float t = dht.readTemperature();
 byte precision = 2;
 dtostrf(t, precision+3, precision, floatBuffer);

 Serial.print("Temperature: ");
 Serial.println(t);
 if (isnan(t)) {
 Serial.println("Failed to read from DHT sensor!");
 return;
 }

 sprintf(msg, "Temp: %s", floatBuffer);
 if((rc = myClient.publish("sensorroom", msg,
strlen(msg), 1, false)) != 0) {
 Serial.println(F("Publish failed!"));
 Serial.println(rc);
 }
```

```
 ...
 }
}
```

3. The following is the code snippet for the Arduino sketch:

```
...
aws_iot_mqtt_client myClient; // init iot_mqtt_client
char msg[32]; // read-write buffer
int cnt = 0; // loop counts
int rc = -100; // return value placeholder
bool success_connect = false; // whether it is
connected

#define DHTTYPE DHT22
#define DHTPIN 10
DHT dht(DHTPIN, DHTTYPE);
...

  // Get a chance to run a callback
 if((rc = myClient.yield()) != 0) {
 Serial.println("Yield failed!");
 Serial.println(rc);
 }
 Serial.print("Sent: ");
 Serial.println(msg);

 delay(3000);
 }
}
```

Now you can open the `aws_iot_config.h` file. You should configure the certificate and private key files, including your AWS IoT endpoint on `AWS_IOT_MQTT_HOST`.

Save this program as `IoTNode`.

Developing a sensor subscriber

To see what sensor data is sent by Arduino Yún , we can create a simple program to subscribe a message on a specific channel. We already define the `sensorroom` channel as the sensor channel. Using the previous program from `Chapter 1`, *Getting Started with AWS IoT,* we can develop a program to subscribe the `sensorroom` channel and then listen to the incoming messages from AWS IoT. We use the Node.js application to communicate with AWS IoT.

Write the following scripts for the complete program:

```
var awsIot = require('aws-iot-device-sdk');

var device = awsIot.device({
 keyPath: 'cert/private.key',
 certPath: 'cert/cert.pem',
 caPath: 'cert/root-CA.pem',
 host: '<host-aws-iot-server>',
 clientId: 'user-testing',
 region: 'ap-southeast-'
 });

device
 .on('connect', function() {
 console.log('connected');
 device.subscribe('sensorroom');
 });

device
 .on('message', function(topic, payload) {
 console.log('recv:', topic, payload.toString());
});

console.log('Sensor subscriber started.');
```

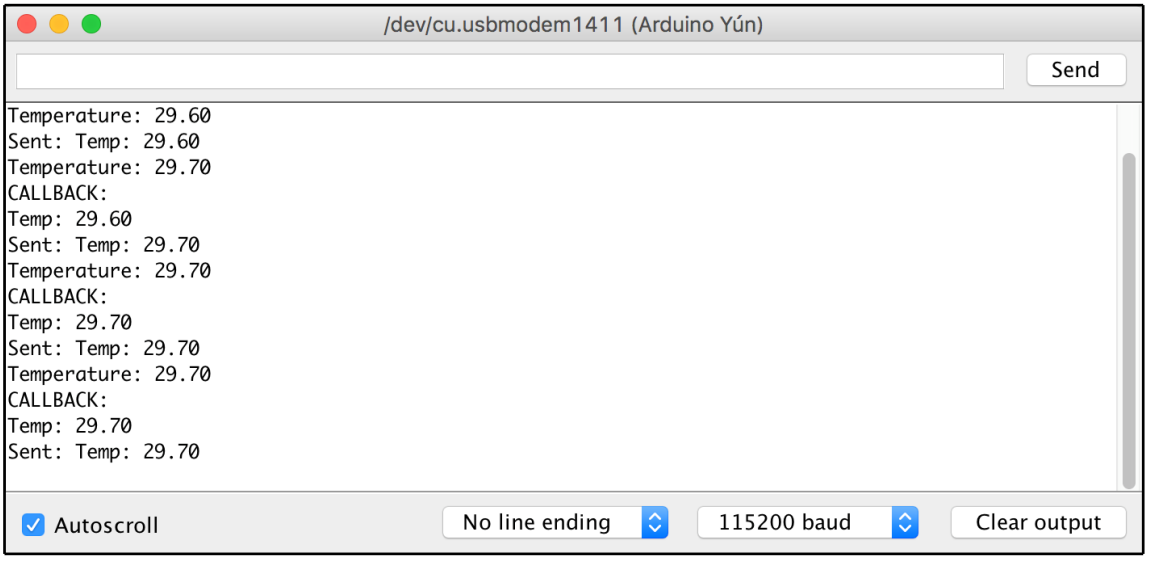
Save this program as the `sensor-subscriber.js` file.

Testing

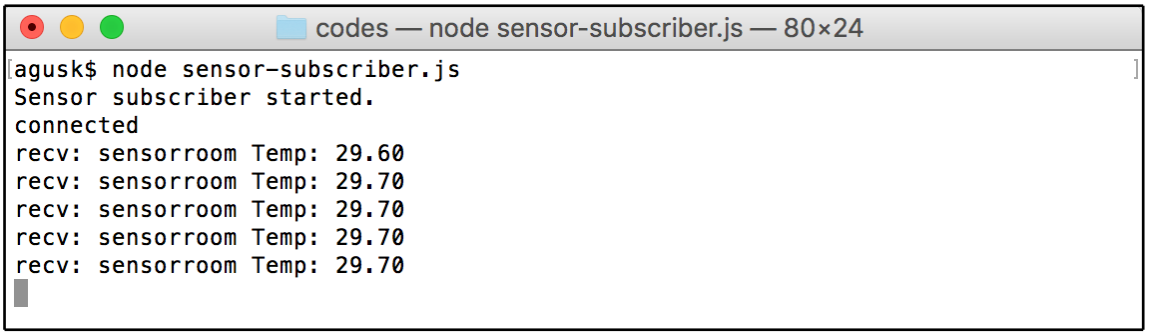
After we deploy the sketch program into Arduino Yún , you can run the sensor subscriber application first. Type the following command on your Terminal or Command Prompt for Windows:

```
$ node sensor-subscriber.js
```

The sketch program will sense `Temperature` and send it to AWS IoT. You can see it on the Serial Monitor Tool. The following screenshot shows `Temperature` from the result of the measurement:



A sensor subscriber application will get sensor data from AWS IoT since it has already subscribed to `sensorroom`. You can see it in the following screenshot:



Summary

We have reviewed some IoT devices and shown how to send data to AWS IoT. Each IoT device has its own AWS SDK to ensure communication with AWS IoT. Finally, we built a simple IoT application to send the sensor data, temperature, to AWS IoT. To read the sensor data, we have a Node.js application to subscribe the sensor channel in order to access the sensor data from AWS IoT.

In the next chapter, we will learn how to build AWS Greengrass and interact with IoT devices.

Hands On - 3
Optimizing IoT Computing Using AWS Greengrass

Some companies probably have security policies that do not allow some internal devices, such as computers and IoT devices, to connect to the internet directly. To solve these issues, they provide proxy or bridge servers to accommodate connectivity between internet devices and internet networks. For IoT cases, we can build a gateway server to serve those requests. In this chapter, we will learn how IoT devices connect to AWS and perform computation locally. We will explore and implement AWS Greengrass to serve all IoT devices to perform AWS computations. We will deploy AWS Greengrass and use Raspberry Pi 3 as AWS Greengrass Core.

The following is a list of topics that we will cover in this chapter:

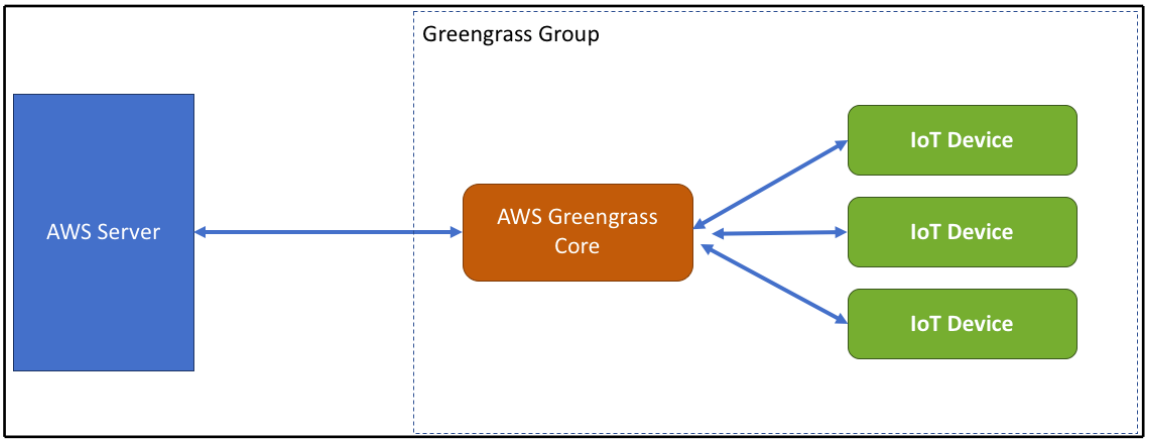
- Introducing AWS Greengrass
- Exploring supported IoT devices for AWS Greengrass
- Deploying AWS Greengrass on Raspberry Pi 3
- Accessing AWS Greengrass
- Building IoT projects with AWS Greengrass

Introducing AWS Greengrass

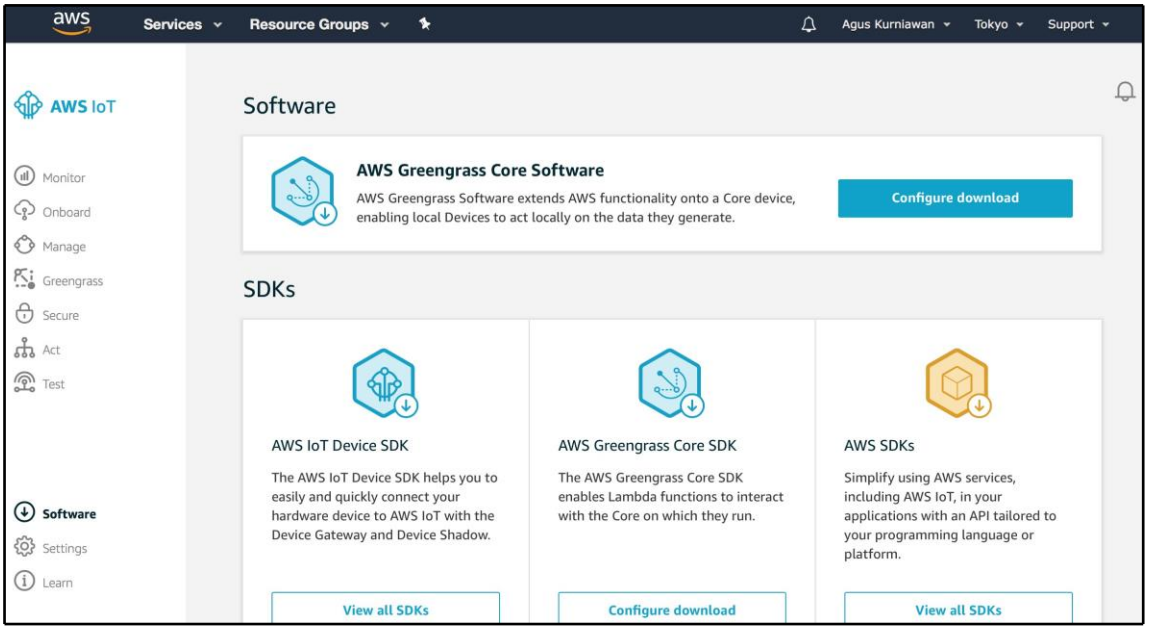
AWS Greengrass is one of the AWS services that enables you to build the AWS IoT server locally. This solution has an advantage, because our IoT devices should not connect to AWS IoT directly.

AWS Greengrass acts as a bridge or gateway for connected IoT devices. It offers AWS computation, such as AWS Lambda. We can deploy our own service over AWS Lambda on AWS Greengrass. AWS Greengrass can be implemented on a computer or device-based ARM CPU. All devices that are used for AWS Greengrass should be registered to AWS IoT. IoT devices have access to AWS Greengrass and AWS Greengrass needs to be configured in one group so that both the IoT devices and AWS Greengrass can communicate with each other.

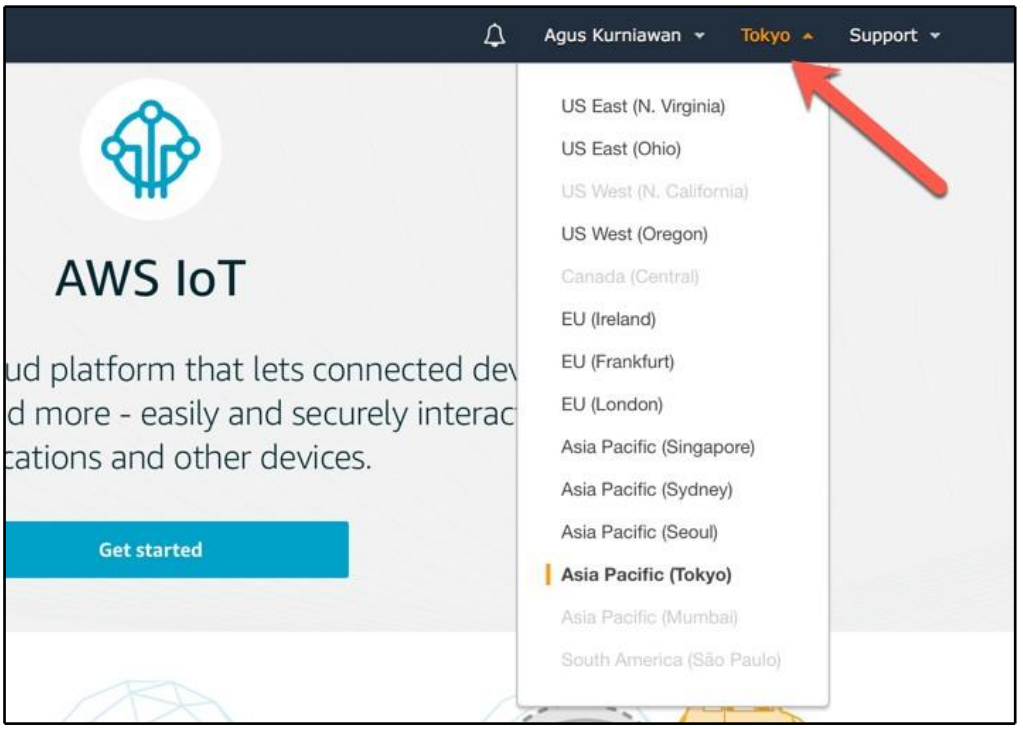
A general overview of AWS Greengrass is depicted in the following figure. Each IoT device can connect to AWS Greengrass to perform computation without connecting to the AWS server. IoT devices and AWS Greengrass can only communicate within the Greengrass Group:



AWS Greengrass Console can be accessed at `https://console.aws.amazon.com/greengrass`. You can review and explore this console. We can manage all AWS Greengrass and IoT devices. The AWS Greengrass console can be seen in the following screenshot:



Not all AWS IoT regions are supported for AWS Greengrass. You can check this while you deploy AWS Greengrass on AWS IoT Management Console. You can change the AWS location on the top menu, as shown in the following screenshot:



Exploring supported IoT devices for AWS Greengrass

To run AWS Greengrass on a machine, we should install AWS Greengrass Core. This software can run on x86_64, ARMv7, and AArch64 (ARMv8) architecture platforms. It also supports Ubuntu 14.04 LTS, Jessie Kernel 4.1/4.4, and other Linux distributions based on Kernel 4.4 or later versions.

The list of supported IoT boards for AWS Greengrass can be found at `https://aws.amazon.com/greengrass/faqs/`. This book will use Raspberry Pi 3 for targeting AWS Greengrass.

Deploying AWS Greengrass on Raspberry Pi 3

In this section, we will deploy AWS Greengrass on Raspberry Pi 3 Model B. To implement AWS Greengrass deployment, we have the Raspberry Pi 3 board and an active AWS account. There are several steps to deploy AWS Greengrass on Raspberry Pi 3. Perform the following steps:

1. Prepare Raspberry Pi 3
2. Configure Raspberry Pi 3 and other IoT devices to AWS IoT
3. Install AWS Greengrass

These steps will be explained in the following sections.

Preparing Raspberry Pi 3

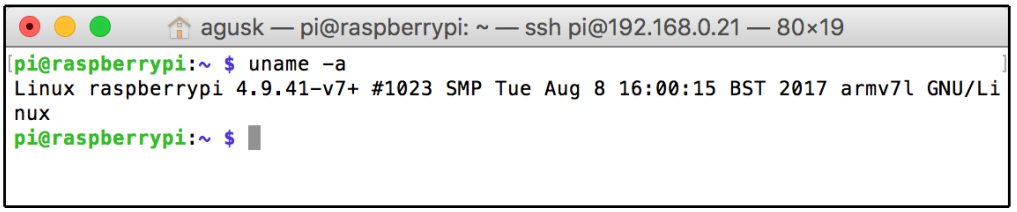
The Raspberry Pi 3 board can run on several OSes. A list of supported OSes for Raspberry Pi 3 can be found at `https://www.raspberrypi.org/downloads/`. To work with AWS Greengrass Core, it is recommended that you install the latest Raspbian Linux OS for Raspberry Pi, which can be found at`https://www.raspberrypi.org/downloads/raspbian/`. For a refresher on installation on Raspberry Pi 3, I recommend you follow the instructions mentioned at `https://www.raspberrypi.org/documentation/installation/installing-images/README.md`.

Now we will prepare to deploy AWS Greengrass Core on Raspberry Pi 3. Follow these steps:

1. You should have installed Raspbian on Raspberry Pi 3. If you have not, you can install it by following the guidelines at `https://www.raspberrypi.org/documentation/installation/installing-images/README.md`. If you have installed Raspbian on Raspberry Pi 3, make sure that the OS has Linux kernel 4.4.11+ or later with OverlayFS and the user namespace enabled. You can verify this using the following command on the Terminal:

```
$ uname -a
```

You should see the kernel version on the Terminal. For instance, you can see my kernel version, which is 4.9.41, from the Raspbian OS in the following screenshot:



2. You also can upgrade Raspbian Linux in order to work with AWS Greengrass. You can type the commands as follows:

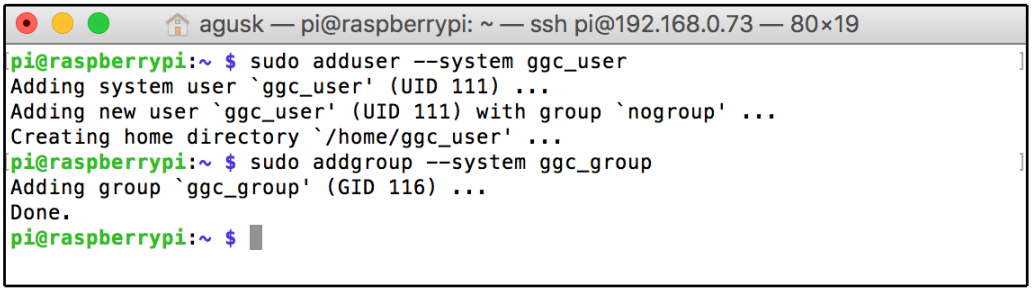
```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo BRANCH=next rpi-update
```

This operation needs internet access, so your Raspberry Pi should be connected to the internet.

3. The next step is to create a user named `ggc_user` and a group named `ggc_group`. Type the following commands on the Raspberry Pi Terminal:

```
$ sudo adduser --system ggc_user
$ sudo addgroup --system ggc_group
```

You can see the executed commands in the following screenshot:



4. AWS Greengrass requires local storage to perform local processing. You can install SQLite on Raspberry Pi. For this, type the following command:

```
$ sudo apt-get install sqlite3
```

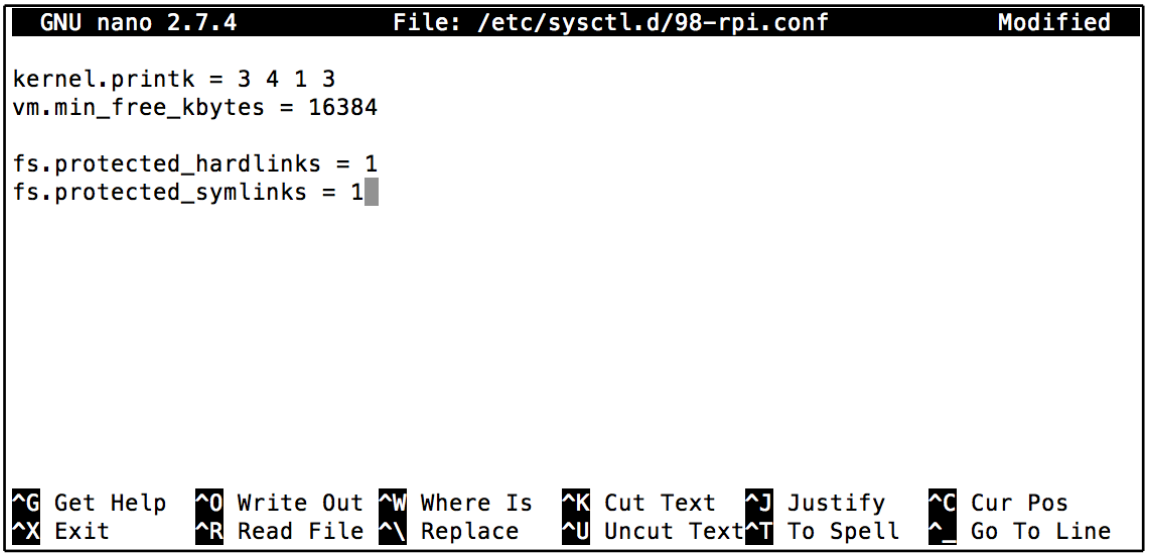
5. Since AWS Greengrass requires a high-security environment, Raspberry Pi should be configured to activate hardlink/softlink protection. You can perform it by opening the `/etc/sysctl.d/98-rpi.conf` file:

```
$ sudo nano /etc/sysctl.d/98-rpi.conf
```

6. Then, add these scripts on that file:

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

You can see this in the following screenshot:



7. Save, and reboot your Raspberry Pi:

```
$ sudo reboot
```

8. After reboot, please type the following command:

```
$ sudo sysctl -a | grep fs
```

You should see our added parameters in the Terminal, as in the following screenshot:

```
fs.nfs.nlm_udpport = 0
fs.nfs.nsm_local_state = 0
fs.nfs.nsm_use_hostnames = 0
fs.nr_open = 1048576
fs.overflowgid = 65534
fs.overflowuid = 65534
fs.pipe-max-size = 1048576
fs.pipe-user-pages-hard = 0
fs.pipe-user-pages-soft = 16384
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
fs.quota.allocated_dquots = 0
fs.quota.cache_hits = 0
fs.quota.drops = 0
fs.quota.free_dquots = 0
fs.quota.lookups = 0
fs.quota.reads = 0
fs.quota.syncs = 0
fs.quota.warnings = 1
```

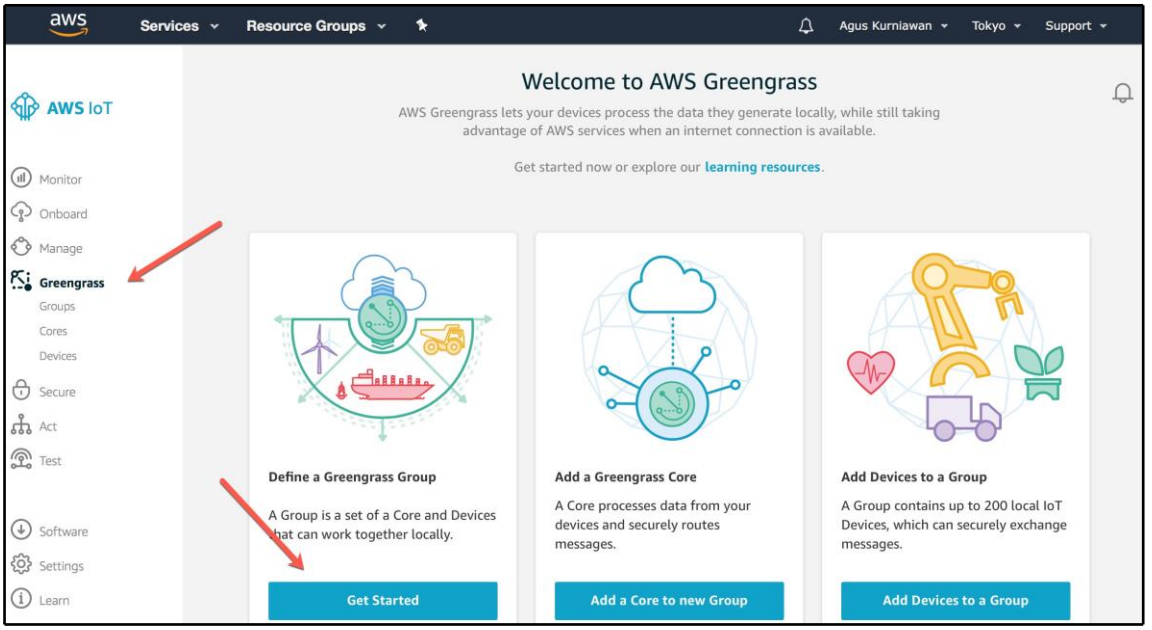
The next step is to configure and register Raspberry Pi to AWS IoT in order to enable AWS Greengrass. We will perform this task in the next section.

Configuring Raspberry Pi 3 and IoT devices to AWS IoT

All devices that are used for AWS Greengrass should be registered to AWS IoT. A result of the registration will be the certificate and private key files that will be used for our program. In this section, we will register our Raspberry Pi 3 to AWS IoT in order to apply for AWS Greengrass.

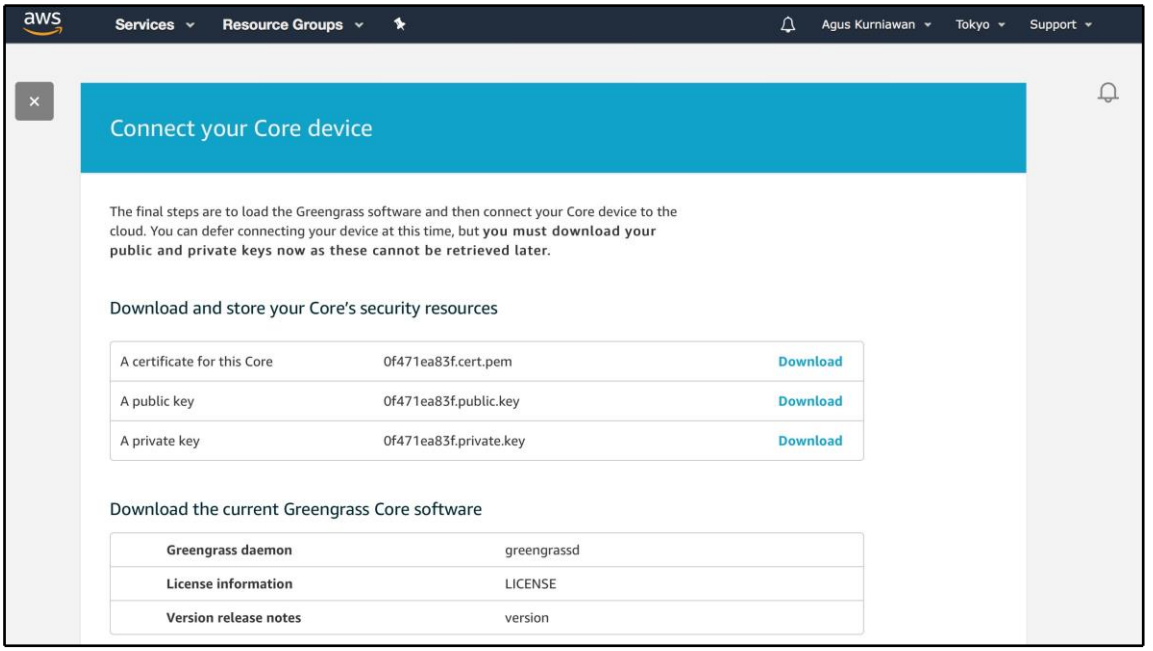
You can follow these steps to configure Raspberry Pi 3 and other IoT devices to AWS IoT Management Console:

1. In AWS Greengrass, from the AWS IoT Management Console panel, start by clicking the **Get Started** button to define a Greengrass Group:

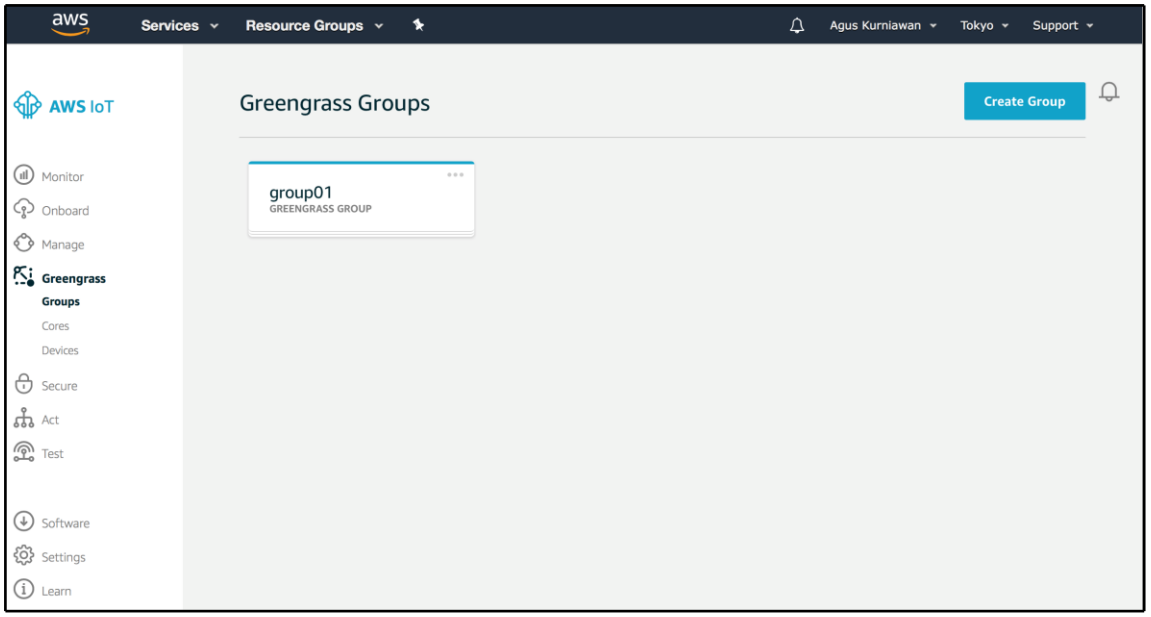


2. Select the **Easy Group creation (recommended)** option. Click on the **Use easy creation** button.
3. Fill in the Greengrass Group name. For instance, `group01`. When done, click on the **Next** button.
4. Then, fill in the Greengrass Core name for our Raspberry Pi 3. For instance, `group01_core`. When finished, click on the **Next** button.

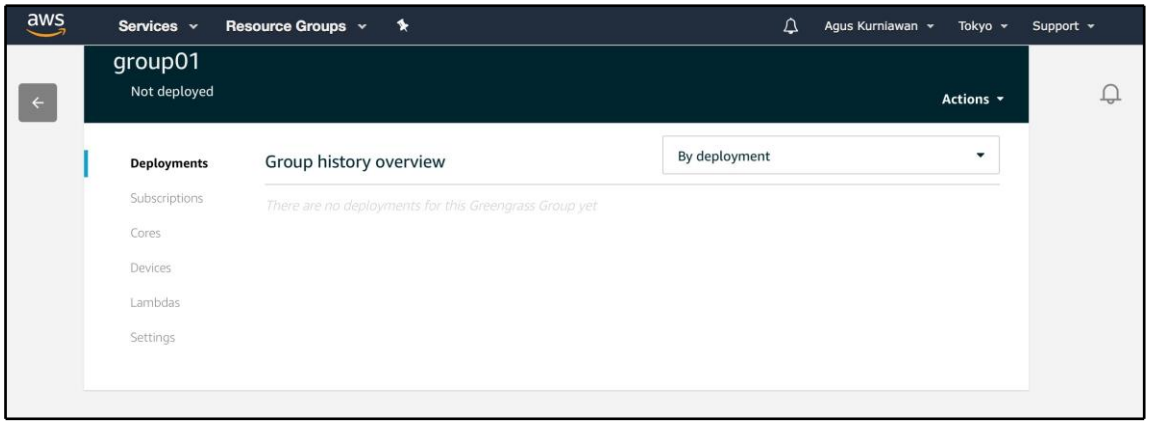
5. You will get a confirmation about creating tasks that will be performed by AWS. If you are ready, you can start to create by clicking on the **Create Group and Core** button. This process will generate security resources for Greengrass Core and Greengrass Core software, such as certificates and key files. You can see them in the following screenshot:



6. Download all files for the certificate, public, and private key files. Also, download Greengrass Core software, which will be deployed into our Raspberry Pi 3 board. Select **ARMv71** for the Raspberry Pi 3 board. Please click on the **Finish** button to complete all the processes.
7. To verify this, you should see our Greengrass name on the **Greengrass Groups** panel, as shown in the following screenshot:



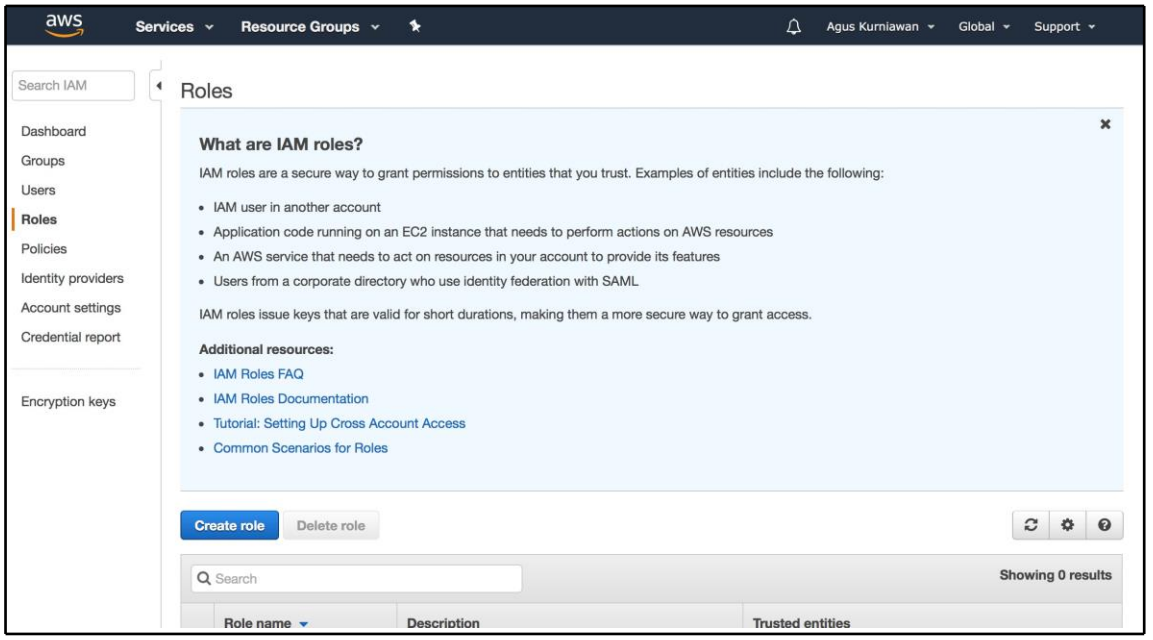
8. If you click your Greengrass Group, you should see the configuration that is shown in the following screenshot:



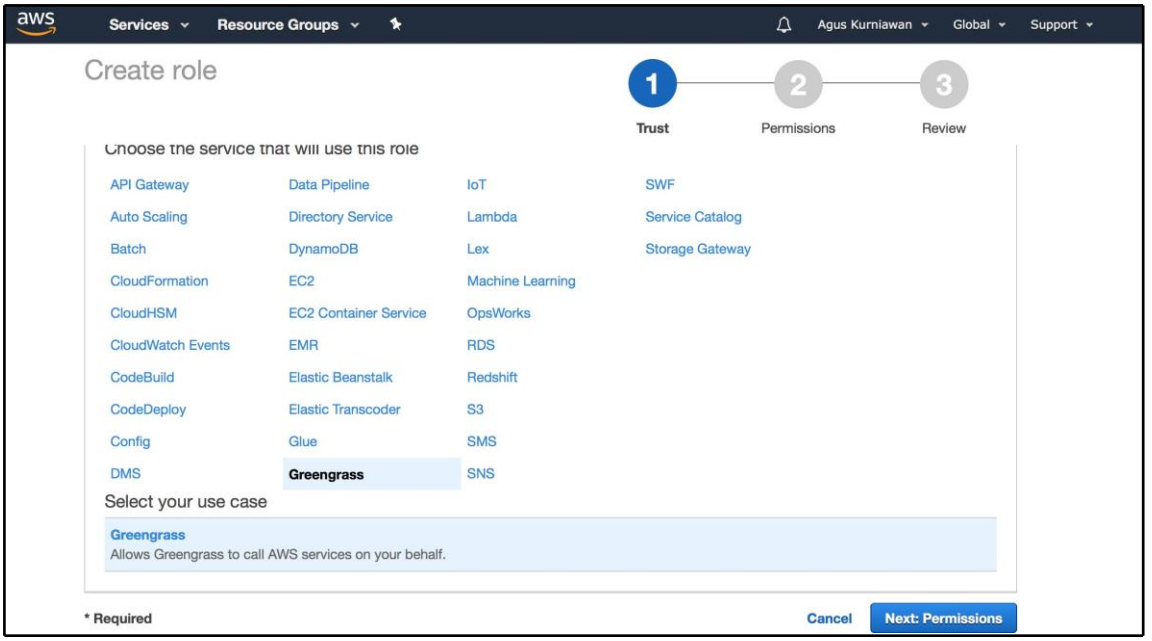
Configuring AWS Greengrass security

By default, AWS Greengrass can be accessed by external systems. We should configure security settings on AWS Greengrass. It's done using AWS IAM. We can create a security role for AWS Greengrass by taking the following steps:

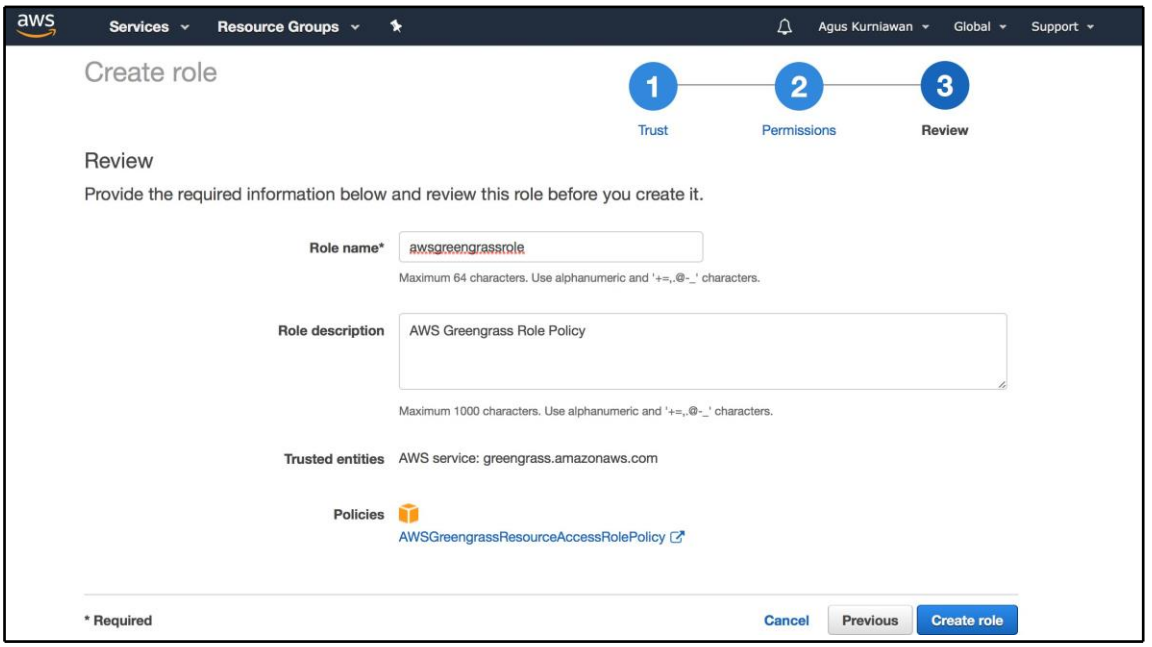
1. Open AWS IAM Console Management on a browser with the URL `https://console.aws.amazon.com/iam`.
2. Create a role by clicking on the **Create role** button from the **Roles** menu, and you will see the following screen:



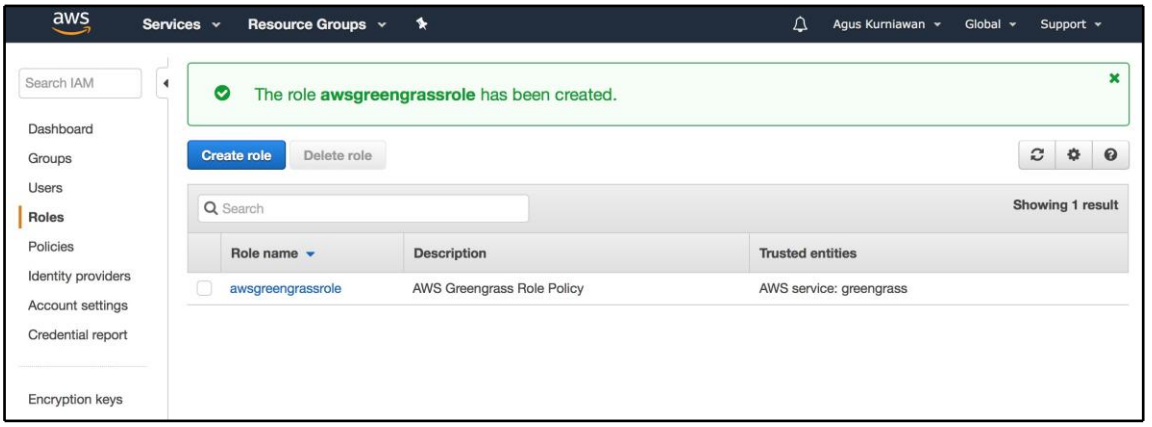
3. Select the **Greengrass** option for the service. When done, click on the **Next: Permissions** button:



4. Fill in the role name. When finished, click on the **Create role** button:



5. After the role is created, you should see it on the **Roles** panel, as shown in the following screenshot:



Installing AWS Greengrass on Raspberry Pi

After we have registered Raspberry Pi 3 to AWS IoT, we can install AWS Greengrass on Raspberry Pi 3. To perform the installation, Raspberry Pi 3 should be connected to the internet. Use the following steps:

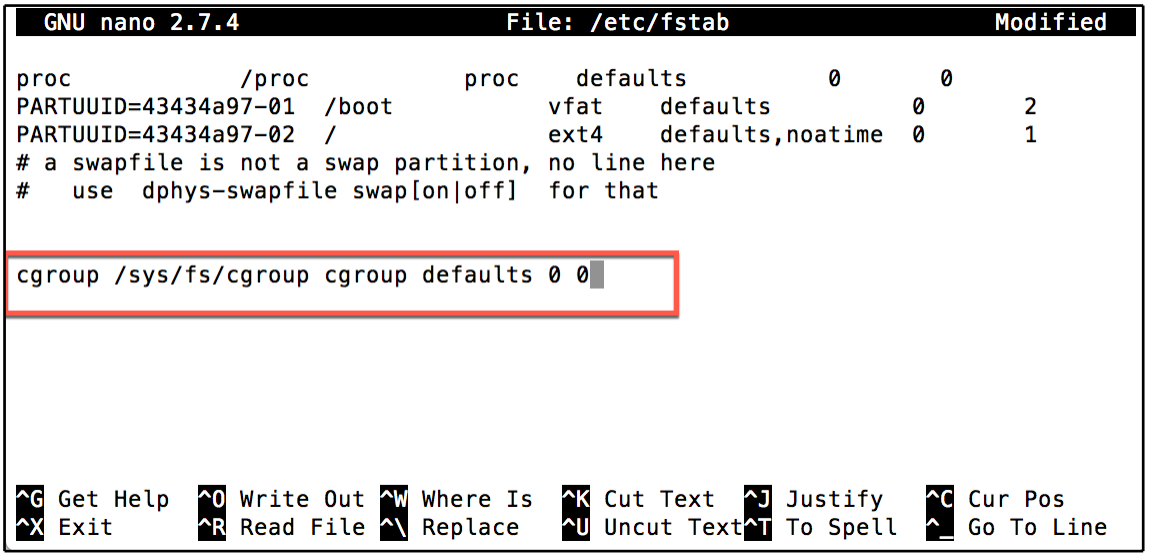
1. Firstly, we configure the Lambda group in order to work with AWS Lambda. Open the `/etc/fstab` file:

```
$ sudo nano /etc/fstab
```

2. Then, add the following script:

```
cgroup /sys/fs/cgroup cgroup defaults 0 0
```

You can see it in the following screenshot:



Save this configuration.

3. In the previous section, we downloaded the Greengrass Core software file. Extract this file into a specific folder.
4. Put all the certificate files into this folder, `<extracted greengrass>/greengrass/certs`. Put the certificate file (`*.crt`), private key file (`*.key`), and AWS root certificate file into this folder. You can download the certificate files from `https://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSign-Class%203-Public-Primary-Certification-Authority-G5.pem`.

5. Then, navigate to the `<extracted greengrass>/greengrass/config` folder. You should see the `config.json` file. If not, you can create the `config.json` file. The following is the content of the `config.json` file:

```
  GNU nano 2.7.4                File: config.json

{
    "coreThing": {
        "caPath": "[ROOT_CA_PEM_HERE]",
        "certPath": "[CLOUD_PEM_CRT_HERE]",
        "keyPath": "[CLOUD_PEM_KEY_HERE]",
        "thingArn": "[THING_ARN_HERE]",
        "iotHost": "[HOST_PREFIX_HERE].iot.[AWS_REGION_HERE].amazonaws.com",
        "ggHost": "greengrass.iot.[AWS_REGION_HERE].amazonaws.com"
    },
    "runtime": {
        "cgroup": {
            "useSystemd": "[yes|no]"
        }
    }
}

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

6. You can fill in the fields as follows:
    - `caPath`: This is an AWS root certificate file (`*.pem`).
    - `certPath`: This is a certificate file from AWS Greengrass Core (`*.crt`).
    - `keyPath`: This is a private key file from AWS Greengrass Core (`*.key`).
    - `thingArn`: This is the ARN from AWS Greengrass Core. You can find it at **AWS IoT** | **Greengrass** | **Cores**. You should see the thing ARN value on that web panel.
    - `iotHost`: This is a hostname from AWS Greengrass Core. Just replace the `[HOST_PREFIX_HERE]` and `[AWS_REGION_HERE]` values from AWS Greengrass Core.

- `ggHost`: This is Greengrass endpoint. You can replace the `[AWS_REGION_HERE]` value for `ggHost`.
- `useSystemd`: Set this field with `yes` to enable `cgroup`.

Save this file.

7. Now you can run the Greengrass Core software. Navigate to the `<extracted greengrass>/greengrass/ggc/packages/<version>` folder. Then, type the following command:

```
$ sudo ./greengrassd start
```

8. If you are successful, you should see the Greengrass daemon being started, as shown in the following screenshot:

```
pi@raspberrypi:~/greengrass/ggc/packages/1.1.0 $ sudo ./greengrassd start
Setting up greengrass daemon
Validating execution environment
Found cgroup subsystem: cpu
Found cgroup subsystem: cpuacct
Found cgroup subsystem: blkio
Found cgroup subsystem: memory
Found cgroup subsystem: devices
Found cgroup subsystem: freezer
Found cgroup subsystem: net_cls

Starting greengrass daemon
Greengrass successfully started with PID: 801
pi@raspberrypi:~/greengrass/ggc/packages/1.1.0 $
```

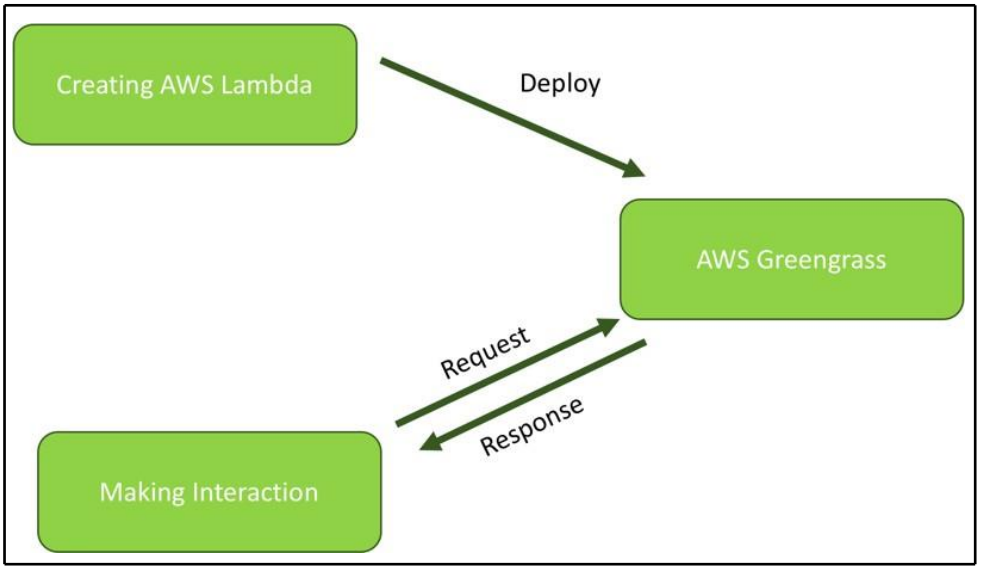
Accessing AWS Greengrass

To access the AWS Greengrass server from IoT devices or other applications, we apply AWS SDK as usual. We only change our target server to the AWS Greengrass machine.

We need to build AWS Lambda into the AWS Greengrass Core machine. Then, we can access it from our IoT devices or computers. We will explore more about AWS Greengrass and IoT device interaction in the next chapter.

Building IoT projects with AWS Greengrass

In this section, we will build a simple scenario to access AWS Greengrass. We will build an AWS Lambda application into AWS Greengrass. Then, we will access it from the client application. Our scenario is illustrated in the following figure:



To implement this demo, we will perform the following tasks:

1. Prepare libraries on AWS Greengrass Core
2. Create AWS Lambda
3. Deploy AWS Lambda to AWS Greengrass
4. Test

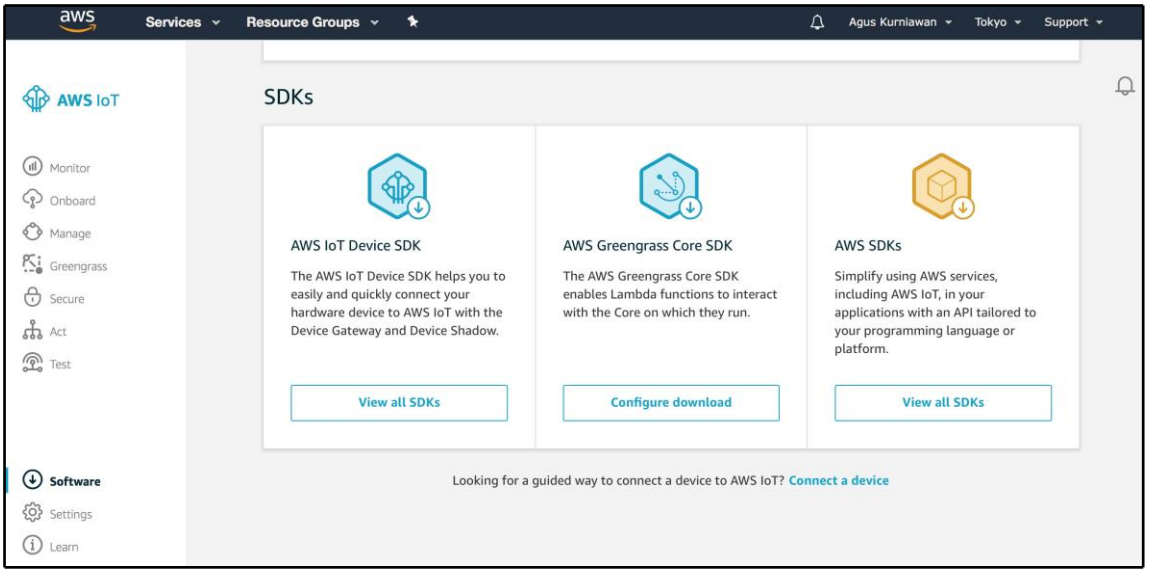
Each task will be explained in the following sections.

Preparing runtime libraries on AWS Greengrass Core

The first task to ensure our AWS Lambda runs well on AWS Greengrass Core is to prepare all requirement modules/libraries. In general, we can define three modules/libraries that should be installed on the AWS Greengrass Core machine. These modules/libraries are as follows:

- AWS IoT SDK
- AWS Greengrass SDK
- External modules/libraries

AWS IoT SDK and AWS Greengrass SDK should be installed on your AWS Greengrass Core machine. You can select a runtime platform, such as Python, Node.js, or Java. Currently, AWS IoT supports Python, Node.js, and Java. You can download these libraries on AWS IoT Console:



You can install the SDKs shown in the preceding screenshot. You can follow the installation guide on the web. Finally, if you use external libraries on AWS Lambda, you should install them on the AWS Greengrass Core machine too. If you have built your own libraries for Python or Node.js, then you can deploy them into Raspberry Pi.

Creating AWS Lambda

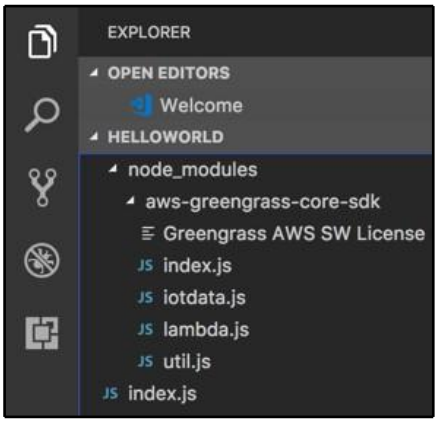
AWS Lambda is an AWS service to help you to build applications for a specific purpose. In this section, we will develop an AWS Lambda application. I will show how to develop AWS Lambda using Node.js.

Firstly, Node.js has to be installed on Raspberry Pi. Then, install AWS Greengrass Core SDK for Node.js. Based on the documentation, we should copy and paste `node` as `nodejs6.10`. You can do this with the following command:

```
$ sudo cp /usr/bin/node /usr/bin/nodejs6.10
```

```
pi@raspberrypi:~ $ node -v
v8.9.1
pi@raspberrypi:~ $ which node
/usr/bin/node
pi@raspberrypi:~ $ sudo cp /usr/bin/node /usr/bin/nodejs6.10
pi@raspberrypi:~ $
```

Prepare the Lambda program sample from AWS Greengrass Core. You can find it at `<greengrass_core_sdk_js>/samples/HELLOWORLD`. Inside this folder, create the `node_modules` folder. Copy the `aws-greengrass-core-sdk` folder from AWS Greengrass Core SDK for Node.js into the `node_modules` folder, as shown in the following screenshot:

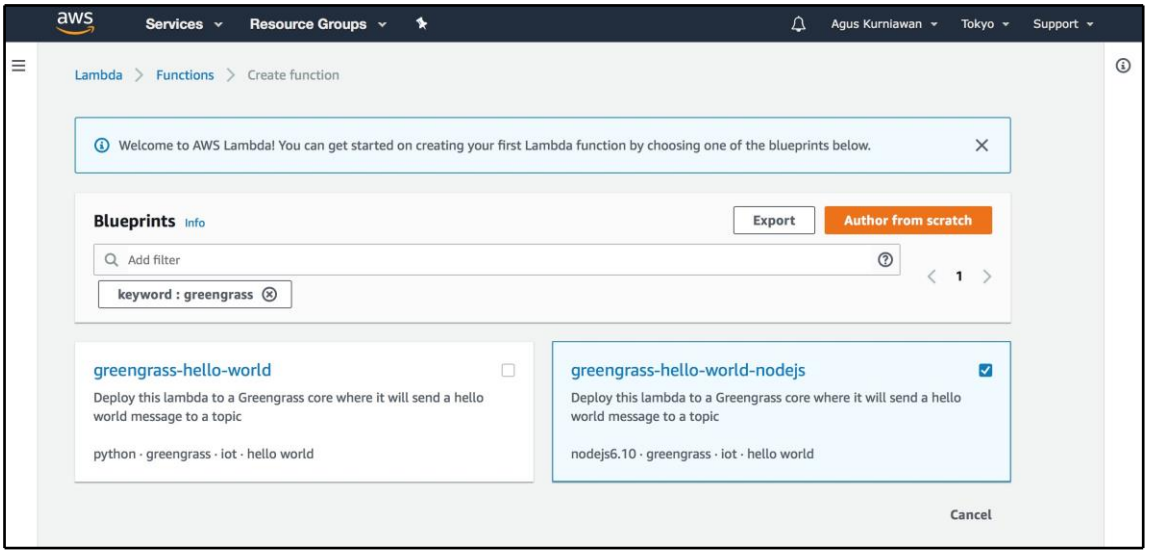


Then, compress all files within the `HELLOWORLD` folder. Make sure `index.js` is the root of the `HELLOWORLD` folder. If not, AWS Lambda will not recognize your Lambda application. Finally, we have the `Helloworld.zip` file that contains the Lambda application. We will upload it on AWS Lambda.

Open AWS Lambda by opening a browser and navigating to `https://console.aws.amazon.com/lambda`. You should see the AWS Lambda dashboard after login.

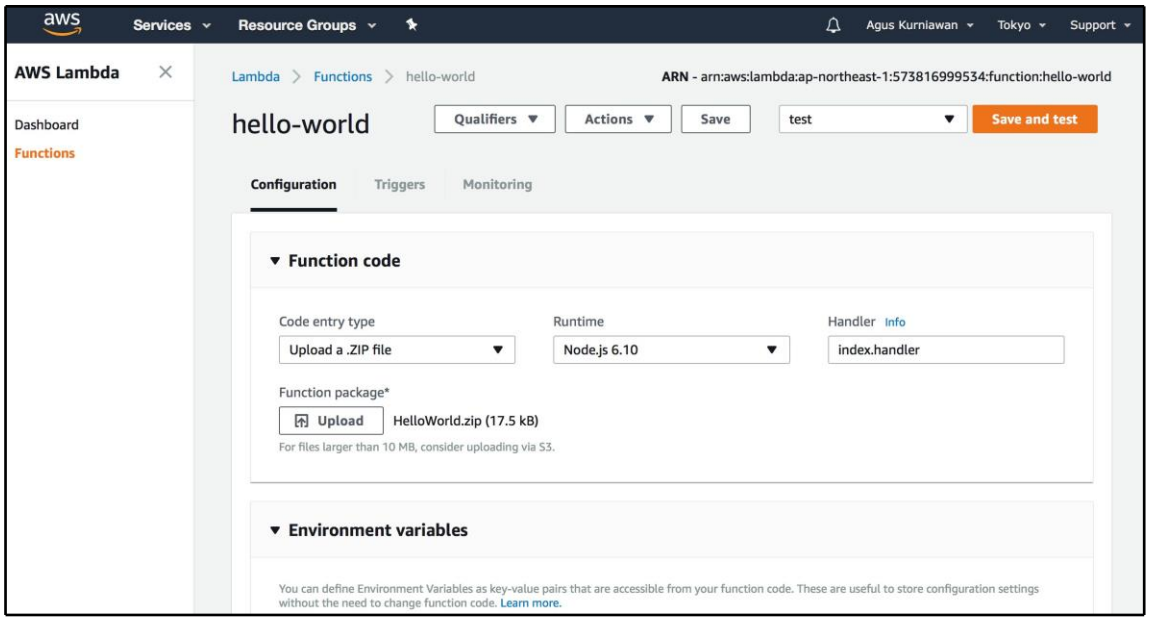
Now you can proceed with the following steps to create your AWS Lambda function:

1. Click on the **Create a function** button. You should see a form, as shown in the following screenshot:

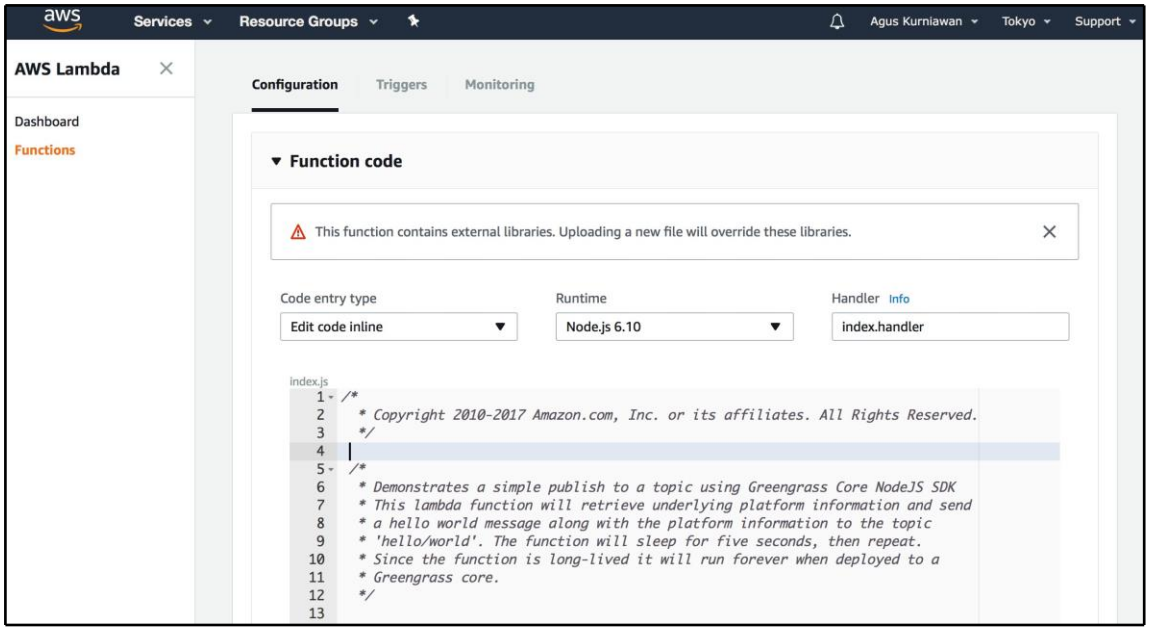


Type `greengrass` so you can see the application samples for Greengrass. Select **greengrass-hello-world-nodejs** by clicking on the checkbox. Then, click on the **Author from scratch** button.

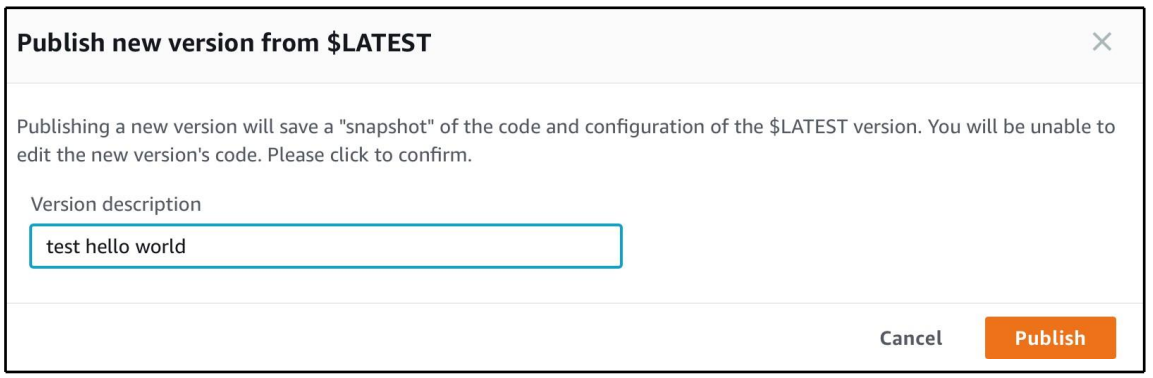
2. Fill in the function and role names. Select the **Create new role from template(s)** option for the role. When done, click on the **Create function** button.
3. Now you should see the Lambda function dashboard. In the **Function code** section, select **Upload a .ZIP file** with runtime **Node.js 6.10** and **index.handler** for the handler name.
4. Click on the `HelloWorld.zip`file that we have prepared to upload the Lambda application:



5. Then, click on the **Save** button to upload and load the Lambda program from the ZIP file. If successful, you should see Lambda program in Node.js. If it has failed, you will probably get error messages while uploading the ZIP file:



6. Now we publish this AWS Lambda. Select **Publish new version** from the **Actions** drop-down list.
7. After selecting, you should be asked to fill in the version description. Once done, click on the **Publish** button to publish AWS Lambda:



The next step is to deploy this service with our AWS Greengrass Core machine.

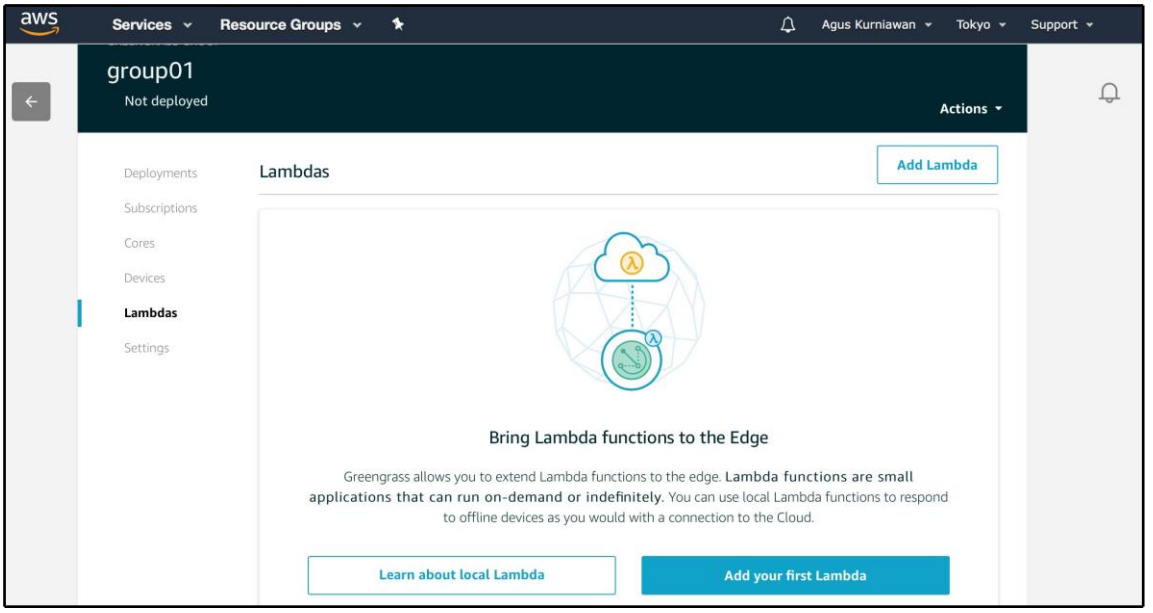
Deploying AWS Lambda with AWS Greengrass

After we have created AWS Lambda, we can continue to deploy AWS Lambda with AWS Greengrass. We perform the following tasks:

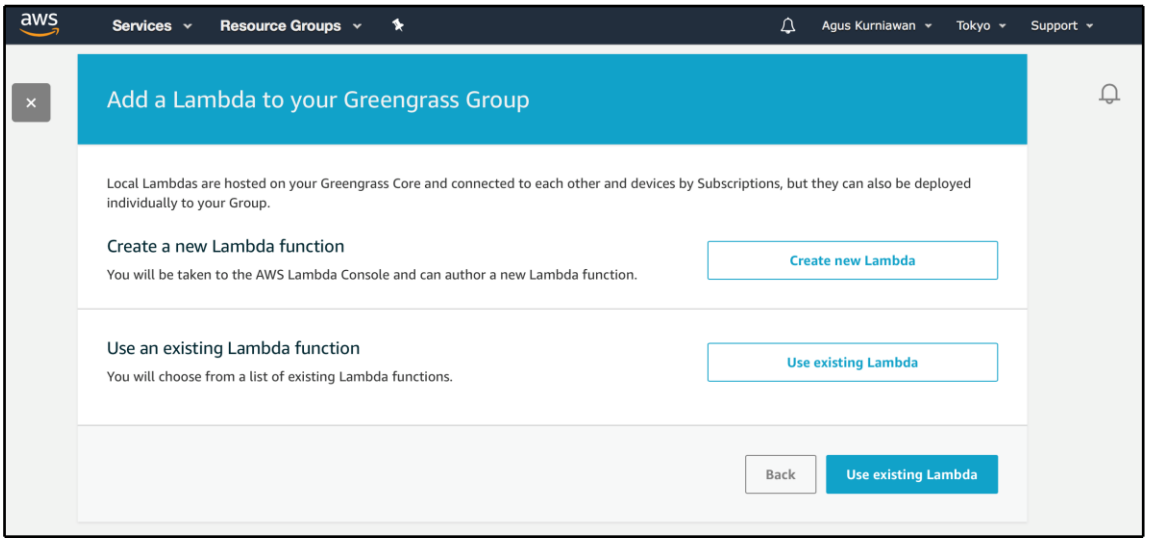
1. Adding Lambda to AWS
2. Configuring Lambda
3. Adding the Lambda subscription
4. Deploying to Greengrass Core

We will perform these tasks in this section:

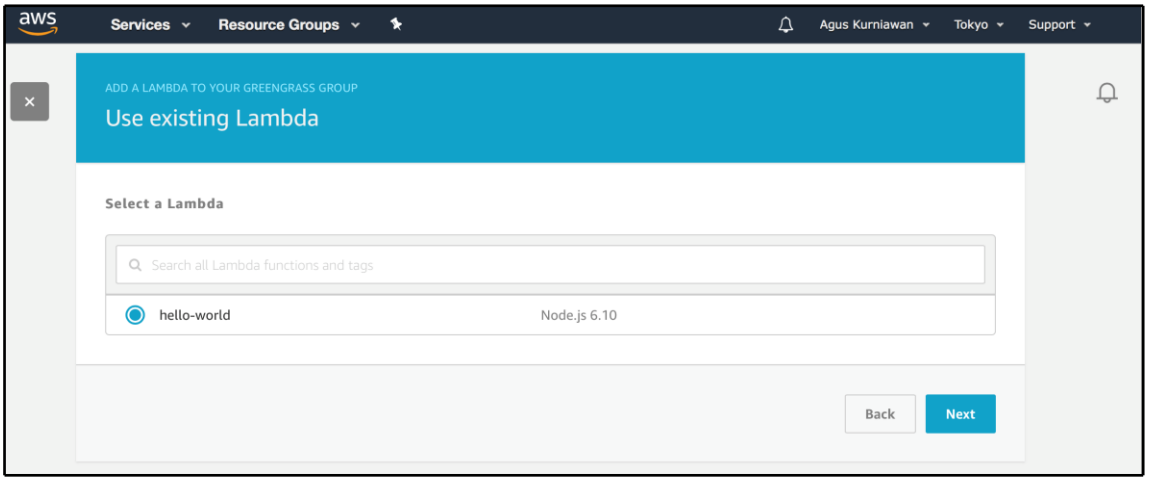
1. Firstly, open your AWS Greengrass Group. Select the **Lambdas** menu that is shown in the following screenshot:



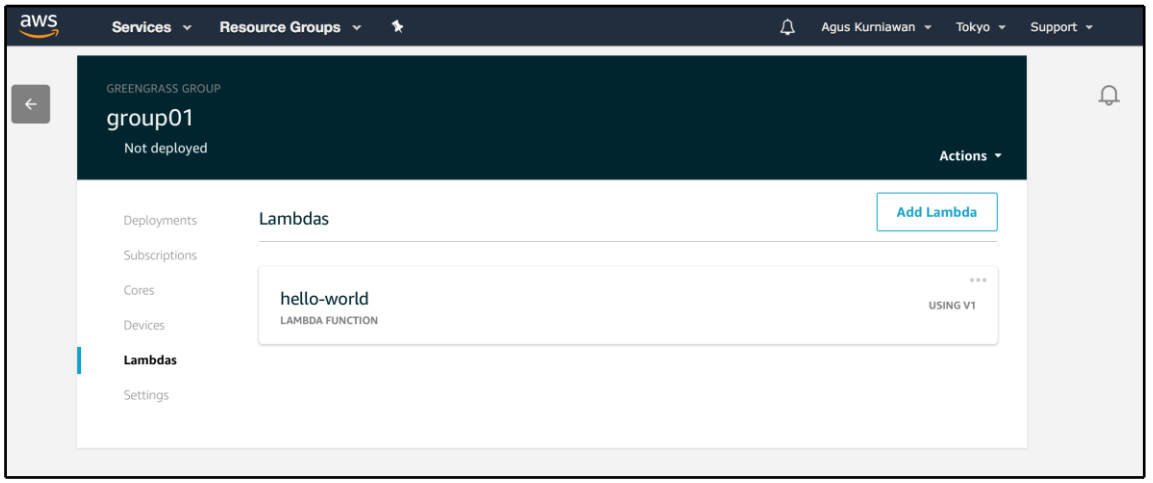
Click on the **Add Lambda** button on the upper-right side. Now you should see the Greengrass Group dashboard, as shown in the following screenshot:



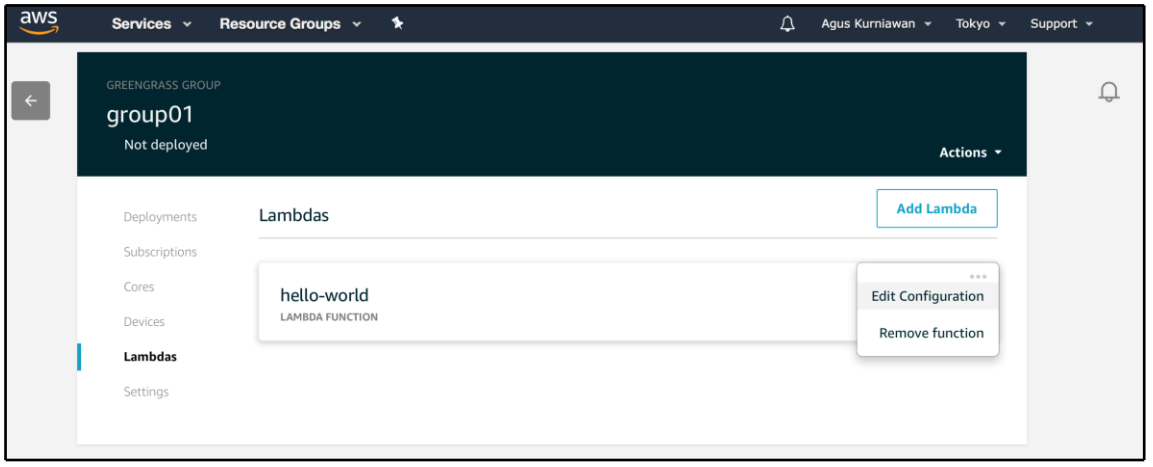
2. Select your created Lambda. Then, click on the **Next** button:



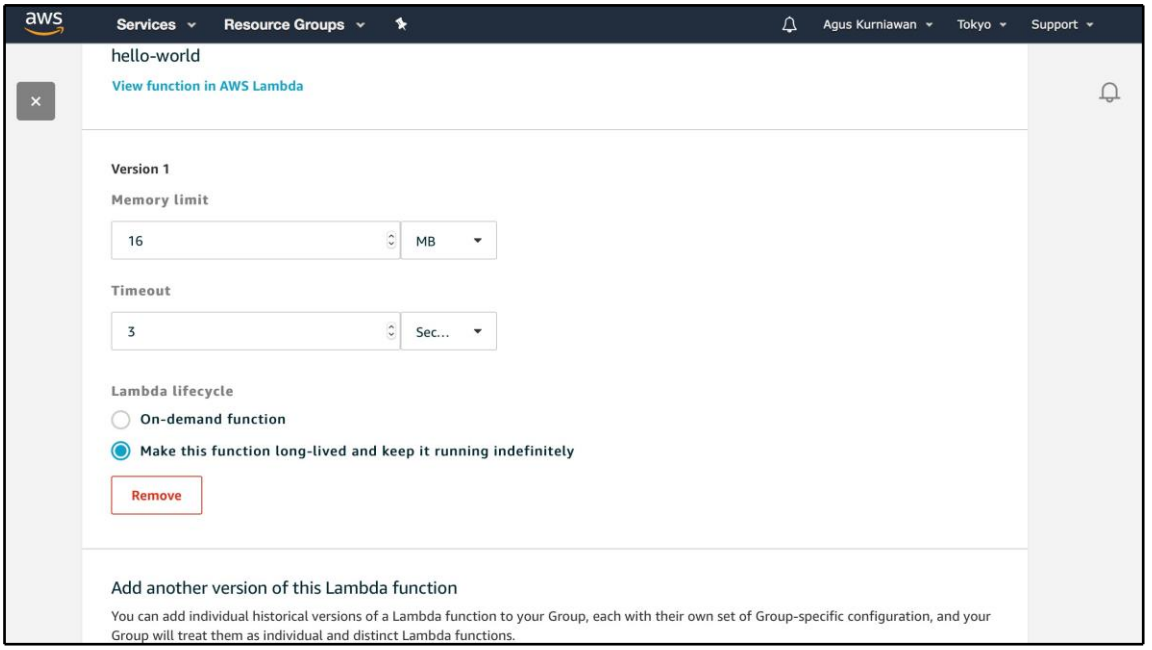
3. You will be asked for the Lambda version. Select it and click on the **Finish** button. If you succeed, you should see your Lambda on the Greengrass Group:



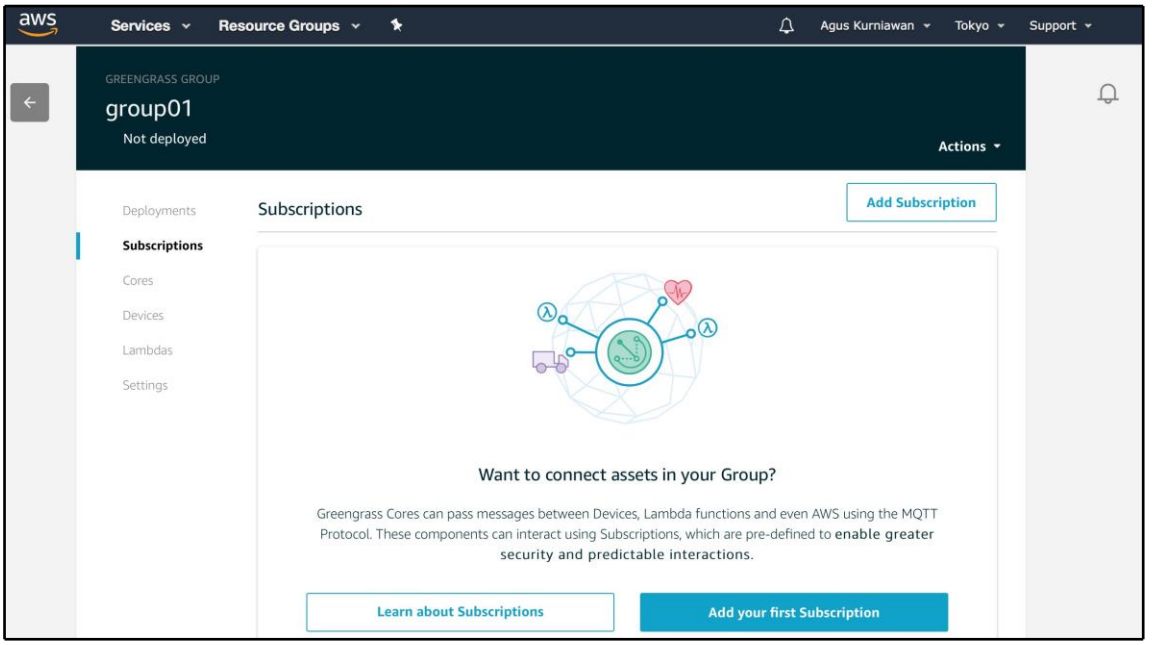
4. Now we configure our Lambda. Click on the ellipsis (**...**) and select the **Edit Configuration** menu, as shown in the following screenshot:



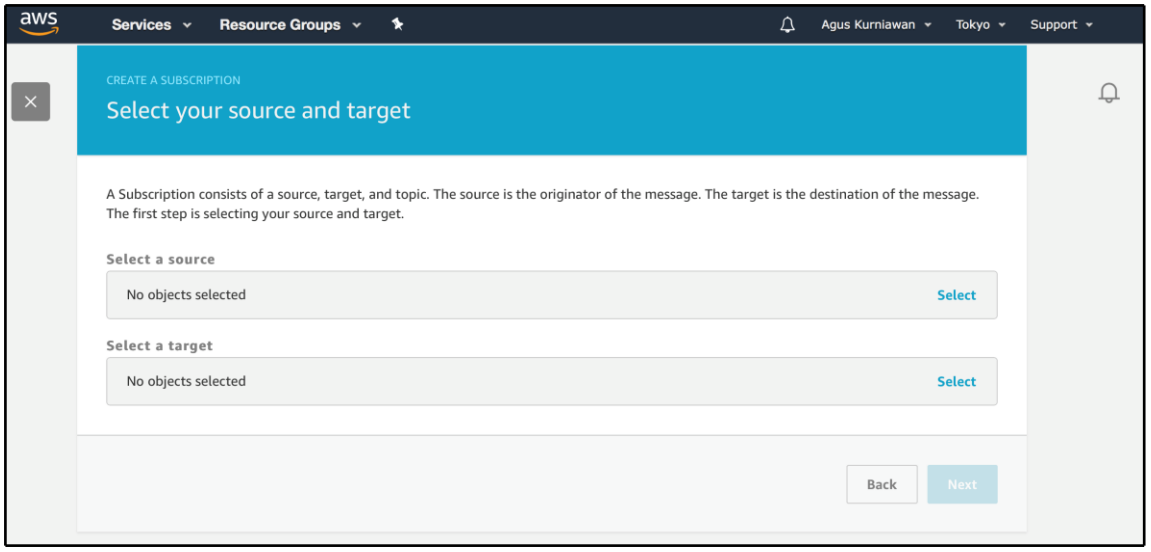
5. Select **Make this function long-lived and keep it running indefinitely** on the radio button. This option makes the Lambda function keep running indefinitely. When done, save all the changes:



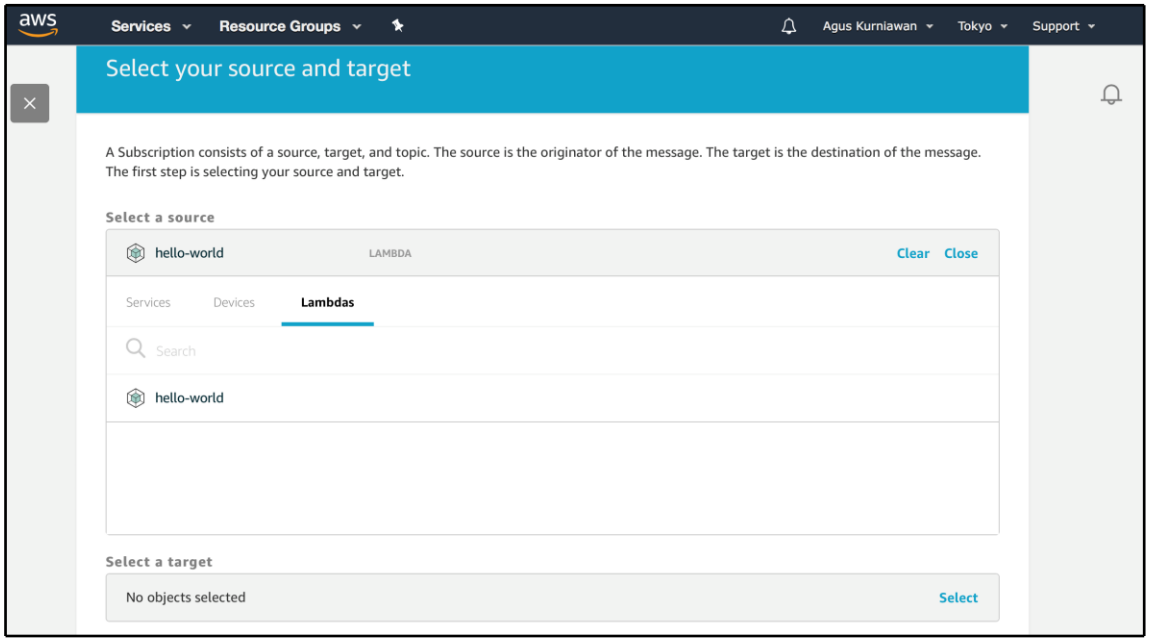
6. Now we add the subscription to access AWS Lambda. Click on the **Subscriptions** menu on the Greengrass Group. You can see it in the following screenshot:



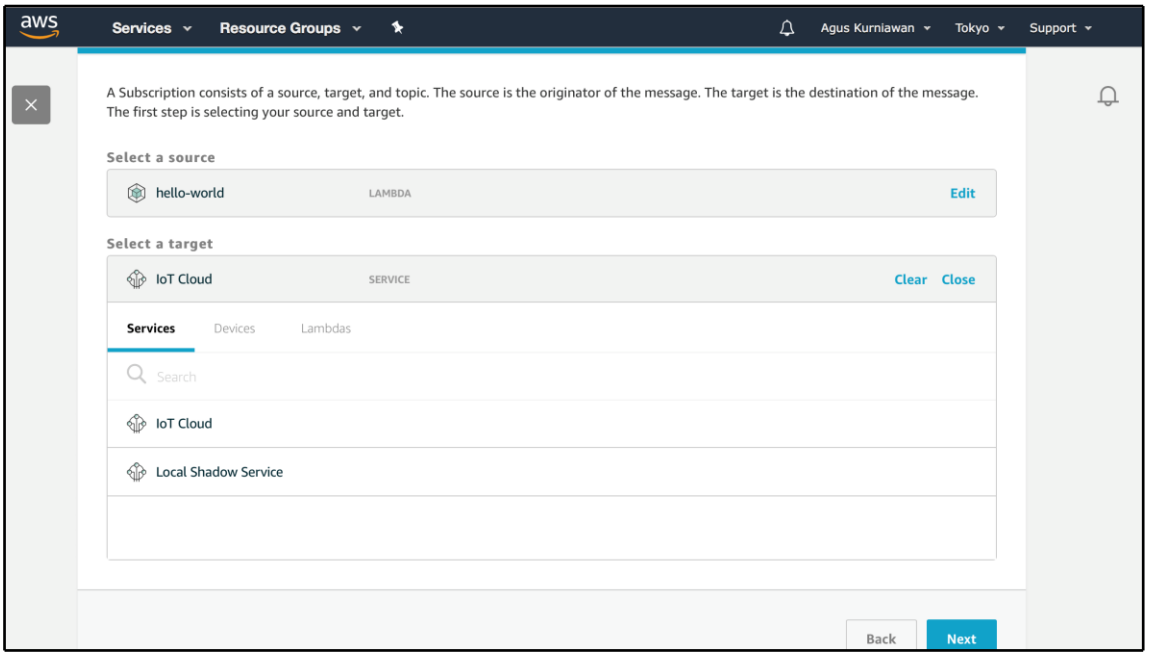
7. Click on the **Add Subscription** button to add a new subscription.
8. Fill in the source on the **Select a source** field and target on the **Select a target** field for our Lambda, as shown in the following screenshot:



9. On the source side, click on the **Lambdas** menu. Then you select your AWS Lambda that has already been created:

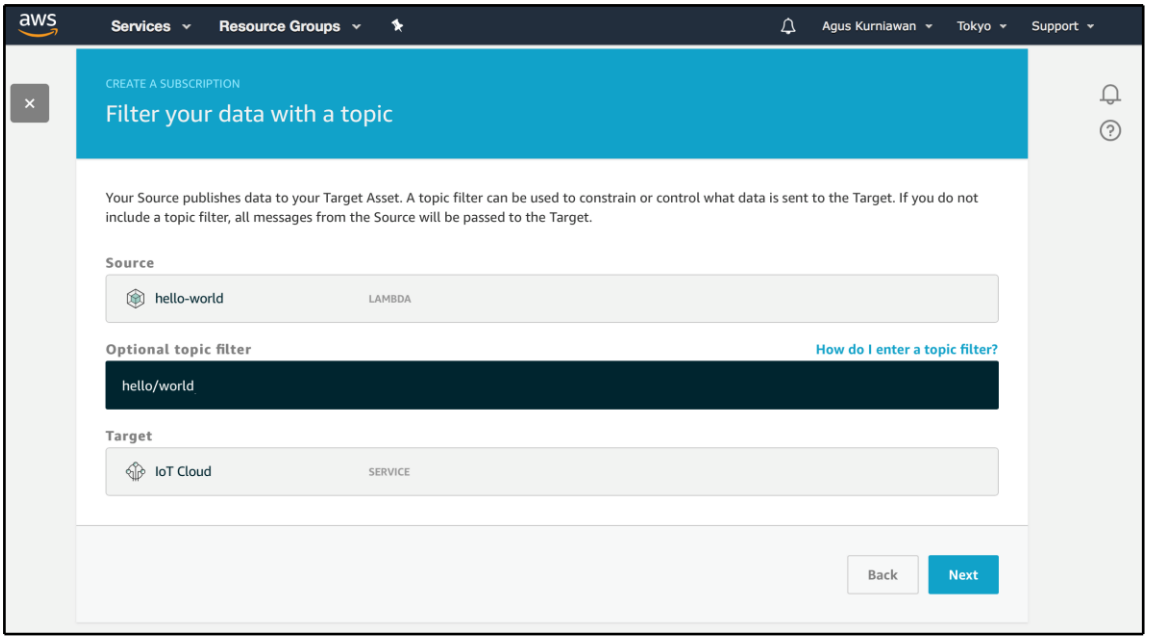


10. On the target side, you click on the **Services** section. Then select the **IoT Cloud** for targeting:



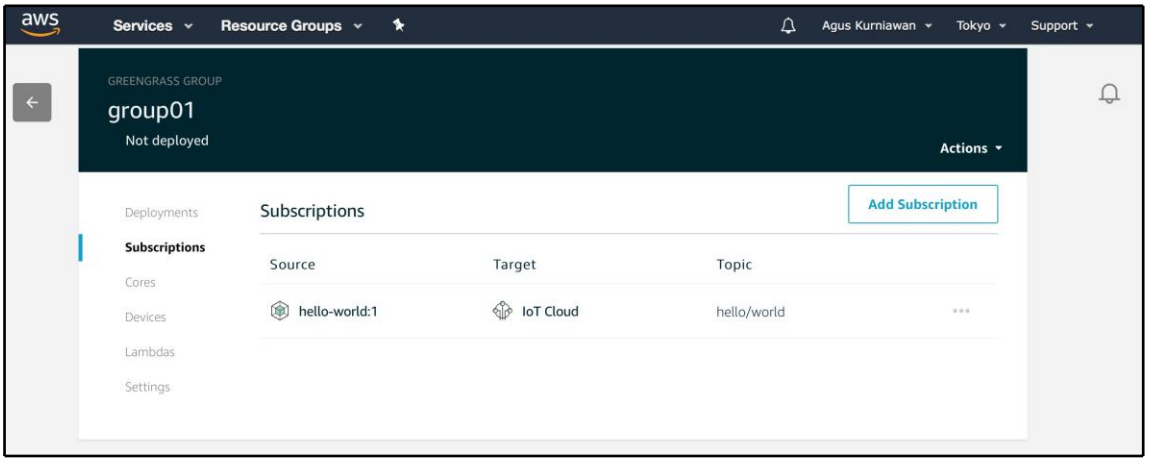
When done, click on the **Next** button.

11. Now we filter our topic. You can filter it on the **Optional topic filter** field. For instance, we set it as `hello/world`:

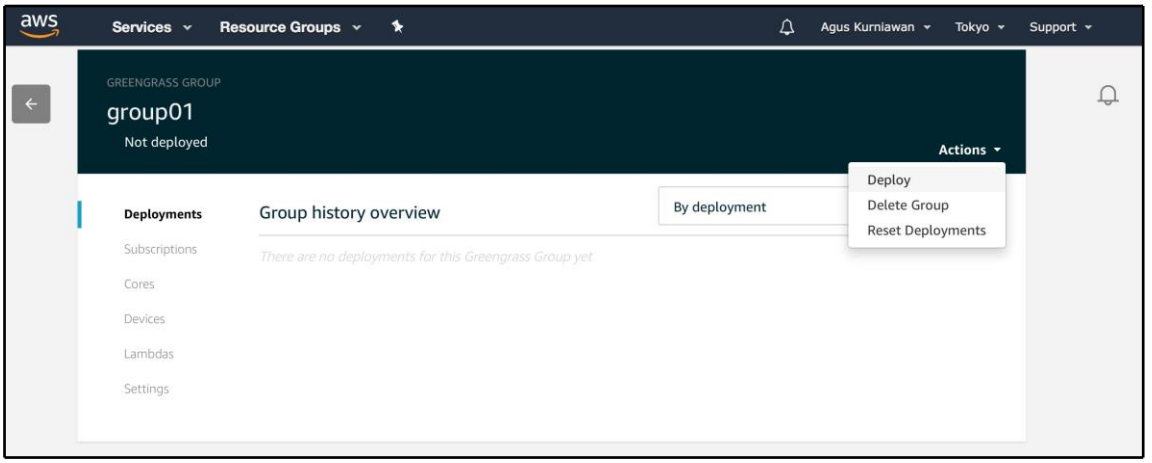


Once done, click on the **Next** button.

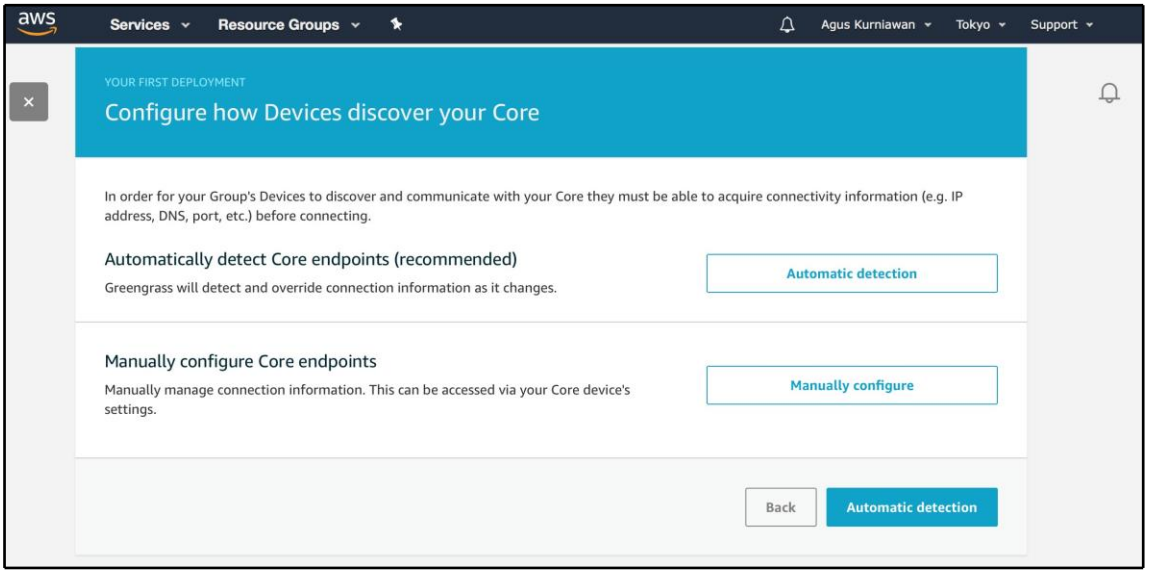
12. After it has been created, you can see your subscription, as shown in the following screenshot:



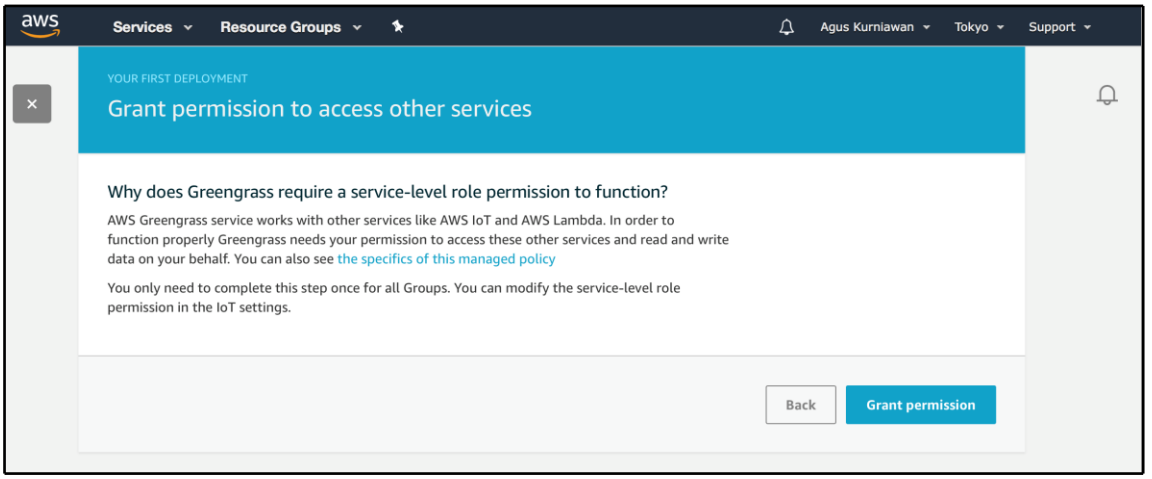
13. The last step is to deploy all the configuration to Greengrass Core. Click on the **Deployments** menu on the Greengrass Group. On the right-side of the menu, click on the **Actions** menu and select the **Deploy** option. You can see it in the following screenshot:



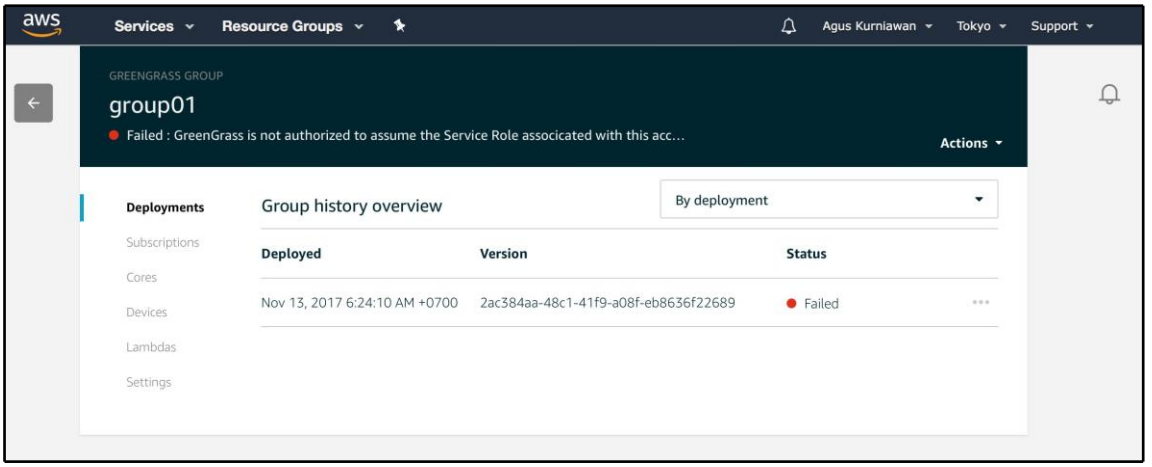
14. You should get the following screen. To simply configure, click on the **Automatic detection** button:



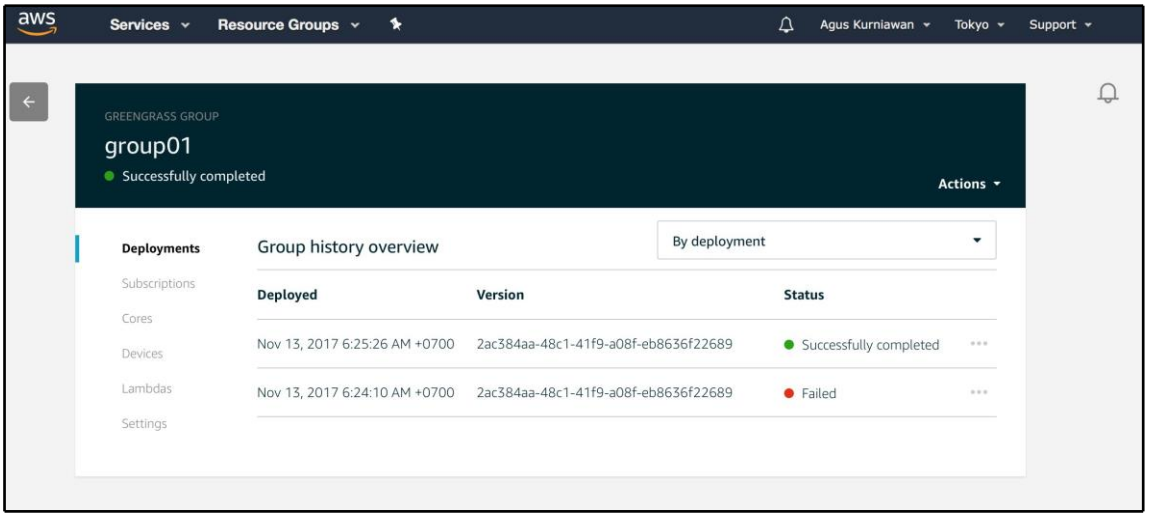
15. Then we grant permissions for other AWS services. Click on the **Grant permission** button:



16. After that, AWS Greengrass will be deployed. You should check the deployment status. For instance, if you get a **Failed** status, as shown in the following screenshot, you should recheck your configuration. Then, try to deploy it again:

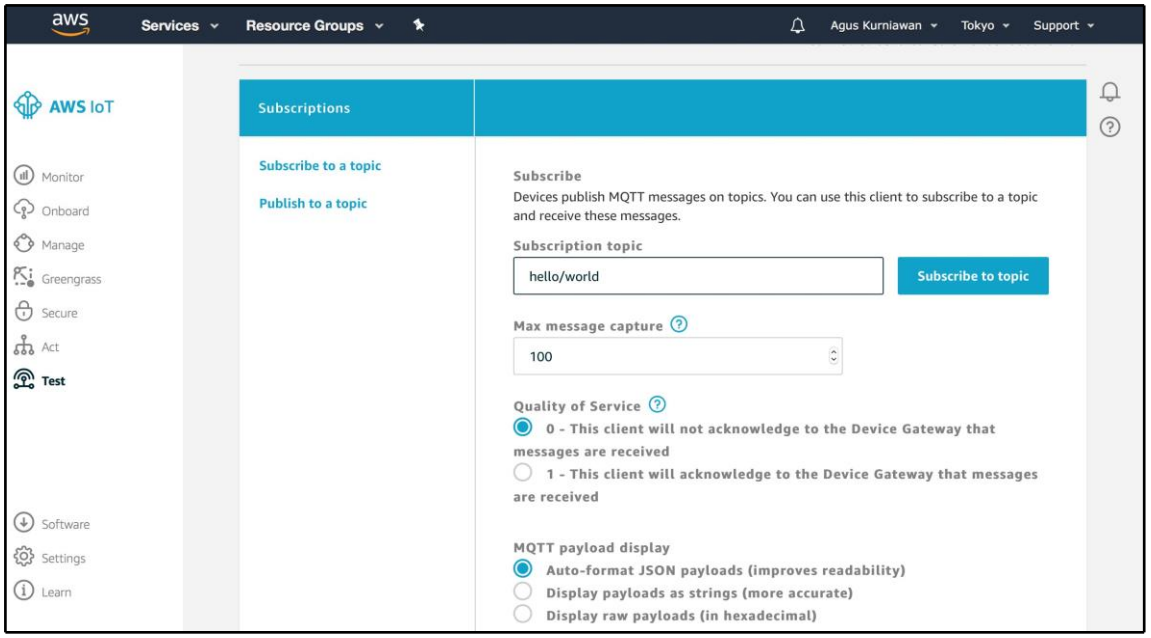


The following is a sample of a successful deployment with Greengrass Core:

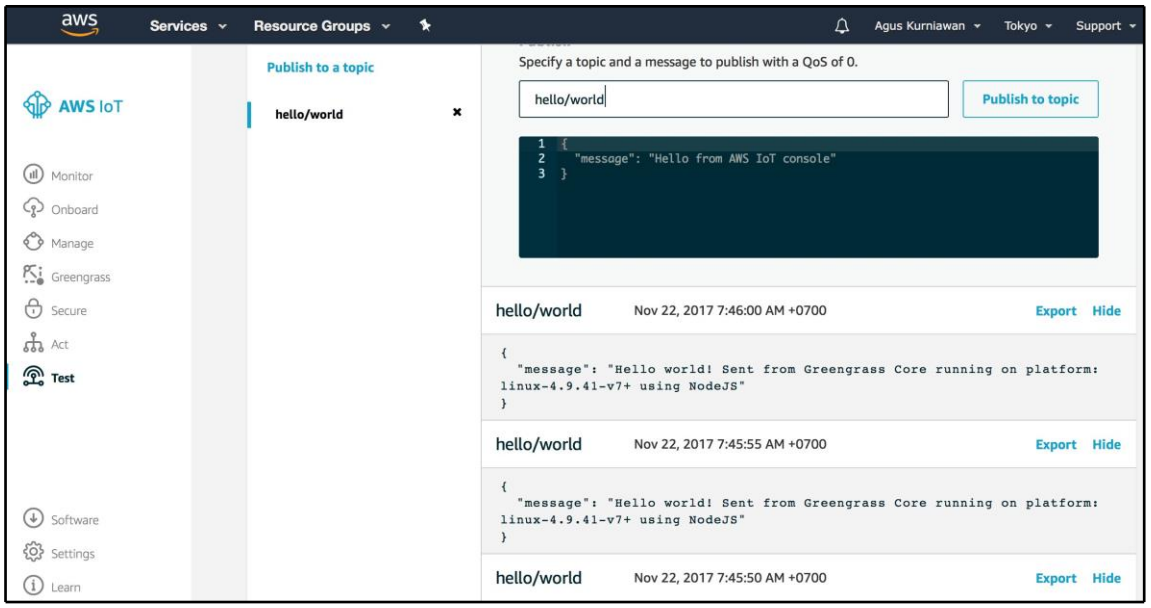


Testing Lambda from Greengrass

We can test our Lambda from Greengrass using a test tool from AWS IoT. Click on the **Test** menu. You should get a form, as shown in the following screenshot. Fill in the subscription topic that we defined while we were deploying Lambda with Greengrass Core:



When done, click on the **Subscribe to topic** button. Now you should see messages from Raspberry Pi, as shown in the following screenshot:



If you do not see messages from Raspberry Pi, you probably have generated errors while configuring or uploading the Lambda application.

Summary

We have learned how to deploy AWS Greengrass on the Raspberry Pi 3 board and make connectivity among IoT devices. We have also built an IoT application to connect AWS Greengrass on Raspberry Pi 3.

In the next chapter, we will learn how to build AWS Lambda on AWS Greengrass and interact with IoT devices.

Hands On - 4
Building Local AWS Lambda with AWS Greengrass

Extending AWS Greengrass functionalities by implementing local AWS Lambda can optimize your resources. In this chapter, we will learn how to implement AWS Lambda to local AWS Greengrass, and how it will be accessed from IoT devices. The following is a list of topics that we will cover in this chapter:

- Introducing AWS Lambda
- Deploying AWS Lambda for AWS Greengrass on Raspberry Pi 3
- Accessing AWS Lambda from IoT devices
- Building IoT projects with AWS Lambda and AWS Greengrass

Introducing AWS Lambda

AWS Lambda is one of the AWS services that provides a serverless compute to serve all the requests when executing your program. Currently, AWS Lambda for AWS Greengrass supports several program runtimes, such as Python, Node.js, and Java.

Implementing AWS Lambda into your IoT project can cut down on your infrastructure costs, especially, the cost of the server availability feature. You can focus on your problems and design your solution.

For further information about AWS Lambda, I recommend that you read the official documentation from Amazon at `http://docs.aws.amazon.com/lambda/latest/dg/welcome.html`.

In this section, I will show you how to develop the AWS Lambda function and then invoke it. We will use the Python program to develop the AWS Lambda function and perform the following steps to implement our demo:

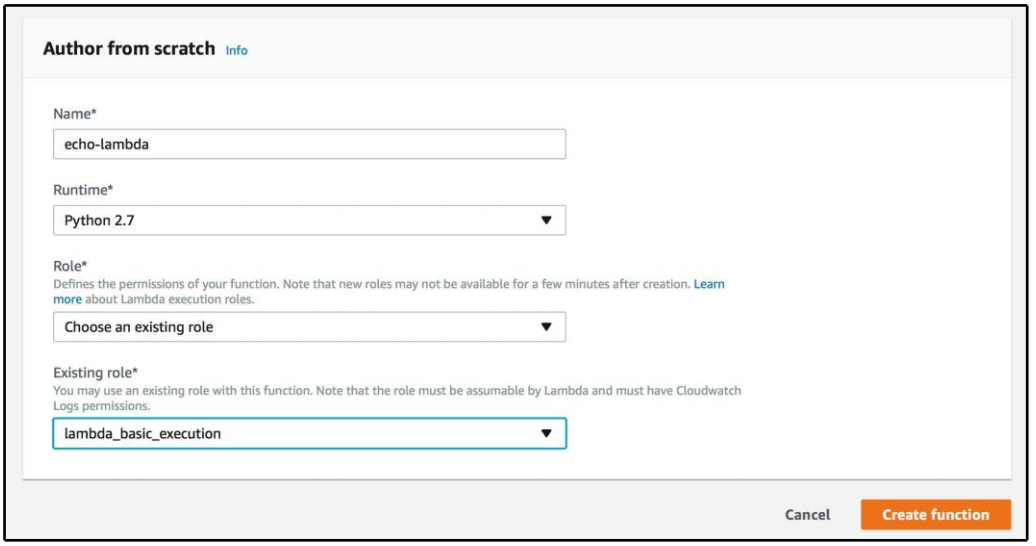
1. Creating the AWS Lambda function
2. Testing the AWS Lambda function
3. Publishing the AWS Lambda function
4. Configuring the AWS Lambda security
5. Invoking the AWS Lambda function

Each task will be explained in the following sections.

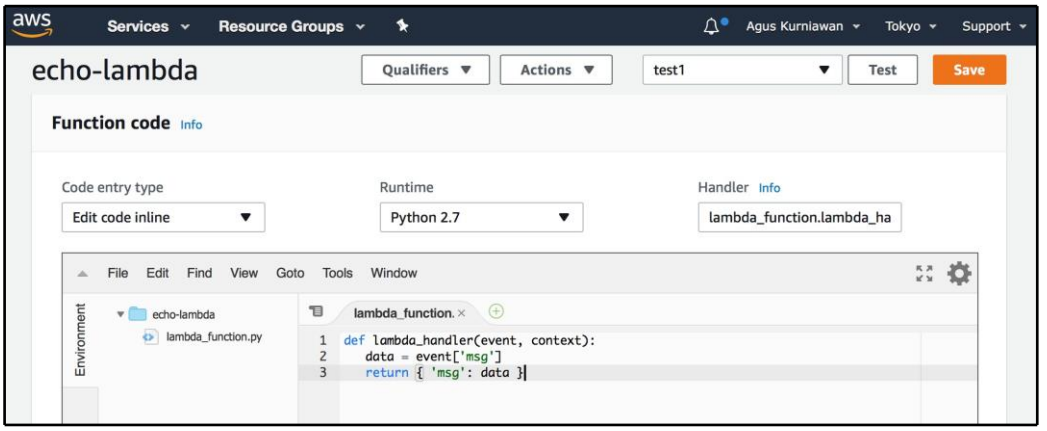
Creating the AWS Lambda function

In this section, we will create the AWS Lambda function by using Python. We will build the AWS Lambda function graphically through a browser. To complete this task, you should have an active AWS account:

1. Firstly, open a browser and navigate to `http://console.aws.amazon.com/lambda`. Our scenario is to build the `echo` function. It will send back any message from the caller. The message format is JSON.
2. On the AWS Lambda function dashboard, you can create a new Lambda function by clicking on the **Create function** button. Then, fill in the function name and its role, as shown in the following screenshot. When done, click on the **Create function** button. For instance, my Lambda function name is `echo-lambda`:



3. If successful, you will see a code editor for the AWS Lambda function. Select **Edit code inline** from the code entry type. Set **Python 2.7** for the runtime and **lambda_function.lambda_handler** for the **Handler** field. You can see this entry in the following screenshot:



4. The following is our Python code for the AWS Lambda function. We return a message with the attribute `msg` from the caller as a JSON message:

```
def lambda_handler(event, context):
   data = event['msg']
   return { 'msg': data }
```

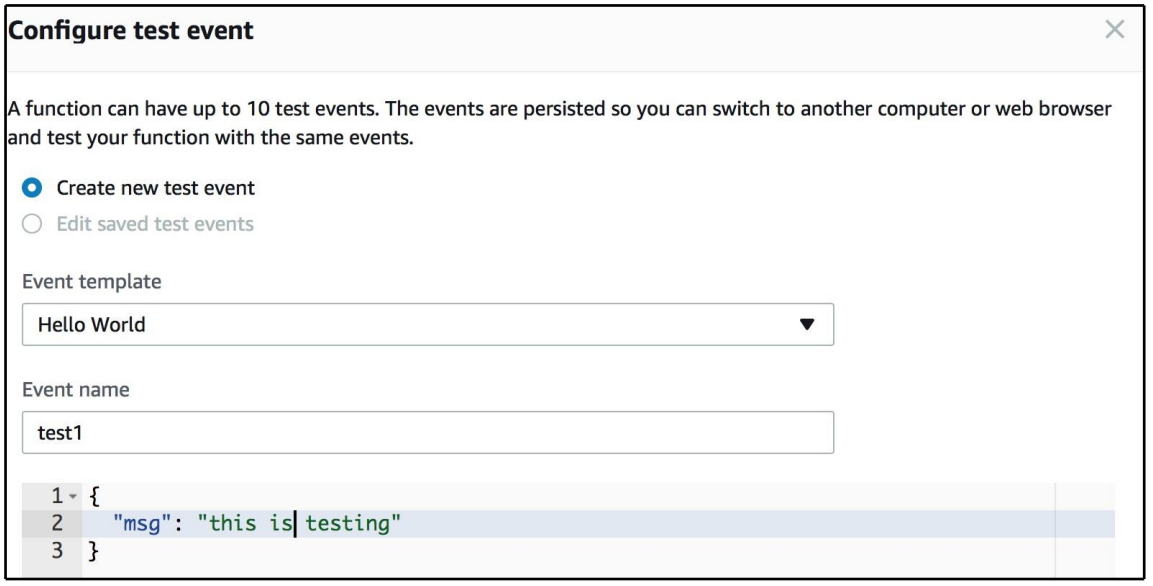
Save these scripts when you are done.

We have created the AWS Lambda. We will test it in the next section.

Testing the AWS Lambda function

After we have created the AWS Lambda function, we should test it before releasing it to the public. Proceed with the steps as following:

1. Click on the **Test** button to create the test scenario and you will get a screen as shown in the following screenshot:

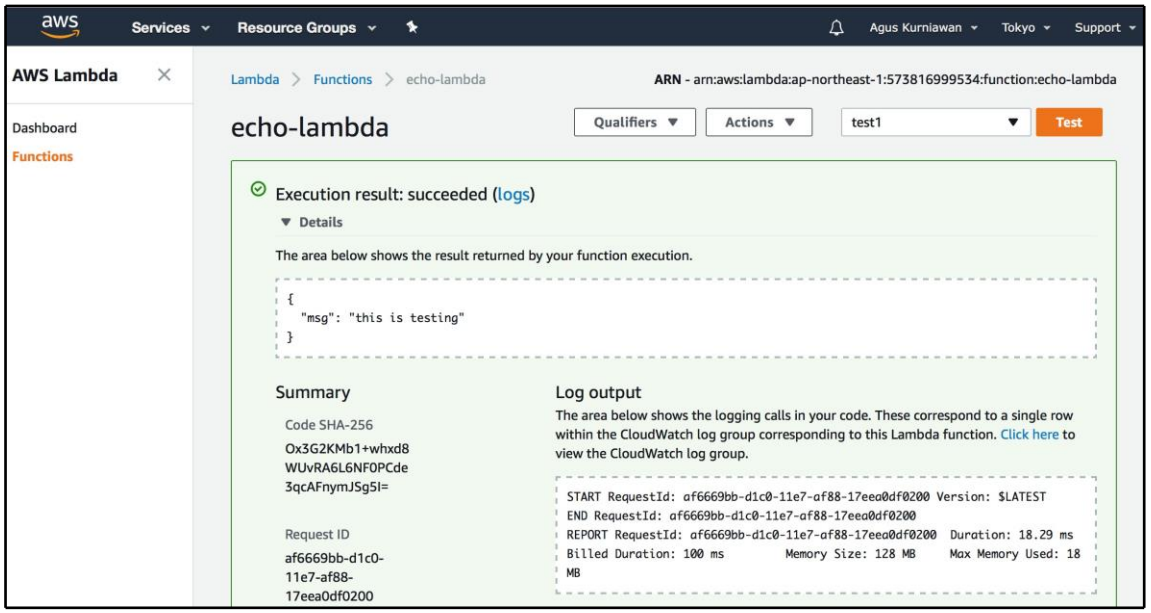


2. For the testing scenario, we create a data sample for testing by writing the following script:

```
{
 "msg": "this is testing"
}
```

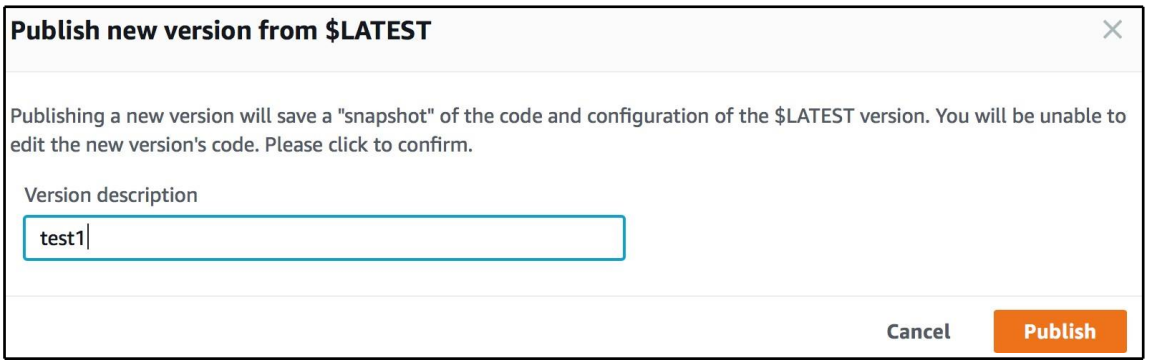
Save this test.

3. Then we try to perform this testing by clicking on the **Test** button. If our testing is successful, we will get a positive response with a green check symbol. You can see my program output in the following screenshot:



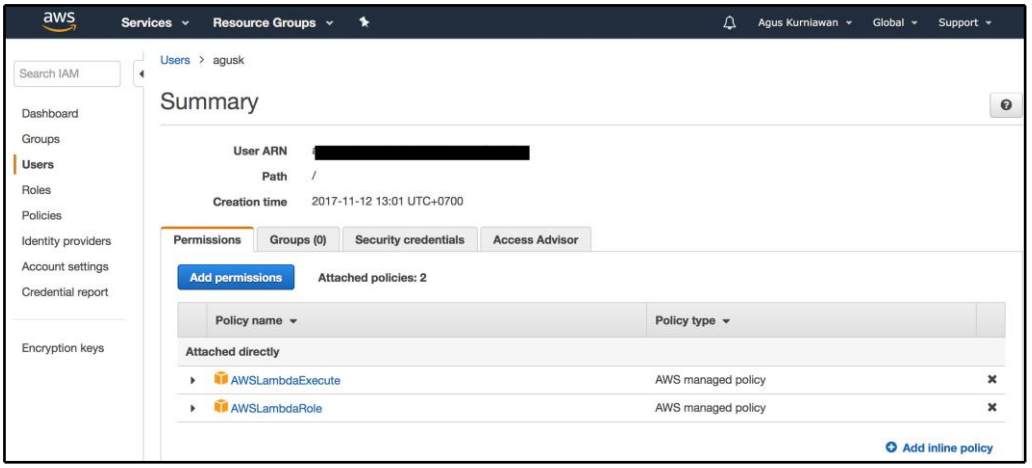
Publishing the AWS Lambda function

Now we can publish our AWS Lambda function. This is done by clicking on the **Publish new version** option on the **Actions** drop-down list. After clicking, you will get a confirmation to fill in the version description. Then, click on the **Publish** button to start publishing the AWS Lambda function:



Configuring AWS Lambda security

Depending on the role setting of the AWS Lambda function, you can configure the AWS Lambda function so that it can be accessed from your account. In this section, we will configure AWS Lambda Security using AWS IAM. You can access AWS IAM through a browser and navigate to `https://console.aws.amazon.com/iam`. Add the **AWSLambdaExecute** permission on your AWS account:



Invoking the AWS Lambda function

We can invoke the AWS Lambda function from our program. For testing, we can execute from the Terminal using the AWS CLI. If you have not installed the AWS CLI yet, you can install it by following the instructions at `http://docs.aws.amazon.com/cli/latest/userguide/installing.html`:

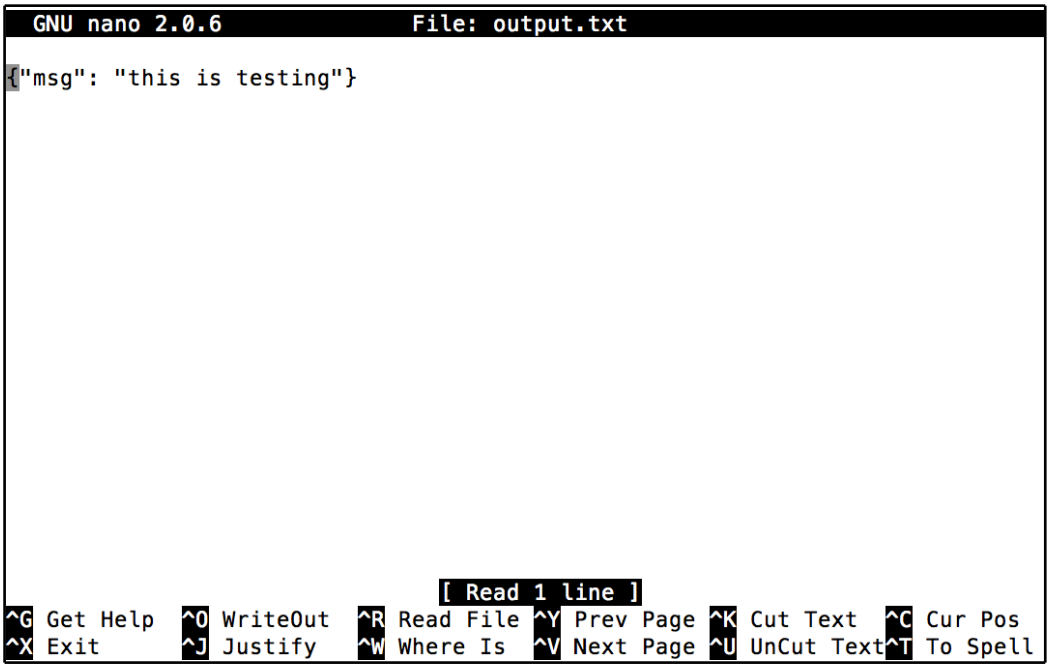
1. For testing, we invoke our AWS Lambda with the payload `{"msg": "this is testing"}`. Type the following command:

```
$ aws lambda invoke --invocation-type RequestResponse --function-
name echo-lambda --payload '{"msg": "this is testing"}' output.txt
```

2. If successful, you should get the return code as `200`, as shown in the following screenshot:

```
agusk$ aws lambda invoke --invocation-type RequestResponse --function-name echo-
lambda --payload '{"msg": "this is testing"}' output.txt
{
    "StatusCode": 200
}
agusk$ nano output.txt
agusk$
```

3. To see the AWS Lambda function returning, we open the `output` file. You should see the message output as follows:



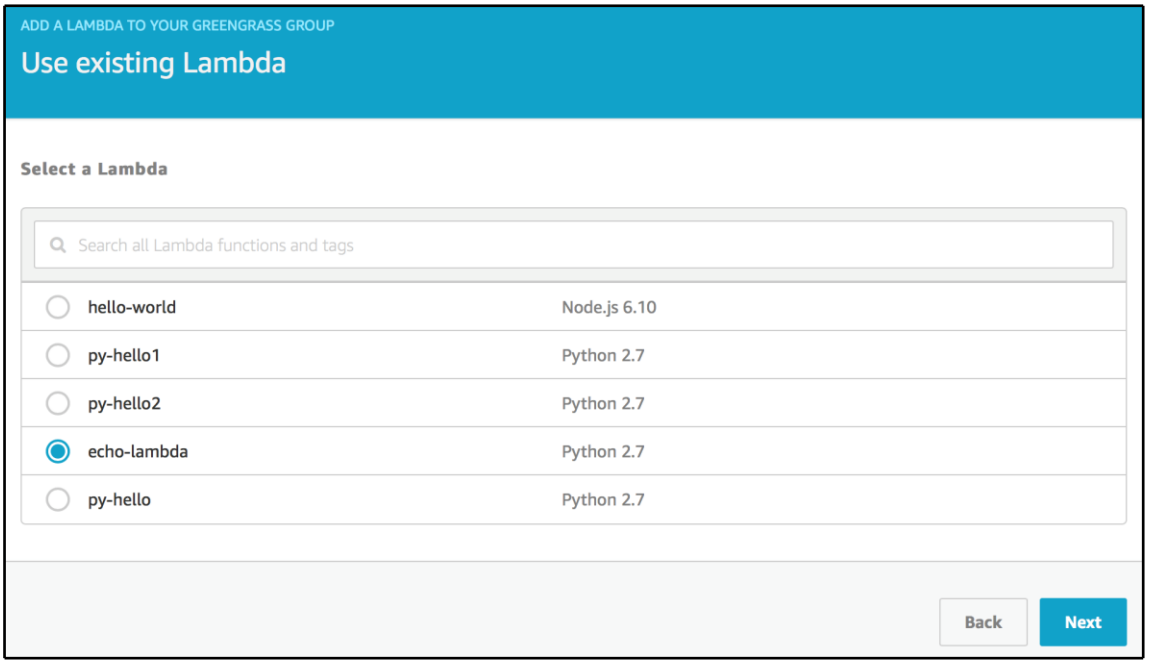
Deploying AWS Lambda with AWS Greengrass on Raspberry Pi 3

In general, we can deploy AWS Lambda with AWS Greengrass graphically and through command lines. In the previous section, we created the AWS Lambda function. In this section, will will deploy AWS Lambda with AWS Greengrass on Raspberry Pi 3.

You have learned how to deploy AWS Greengrass on Raspberry Pi in `Chapter 3`, *Optimizing IoT Computing Using AWS Greengrass*. Please follow the instructions in that chapter to deploy AWS Greengrass Core on Raspberry Pi.

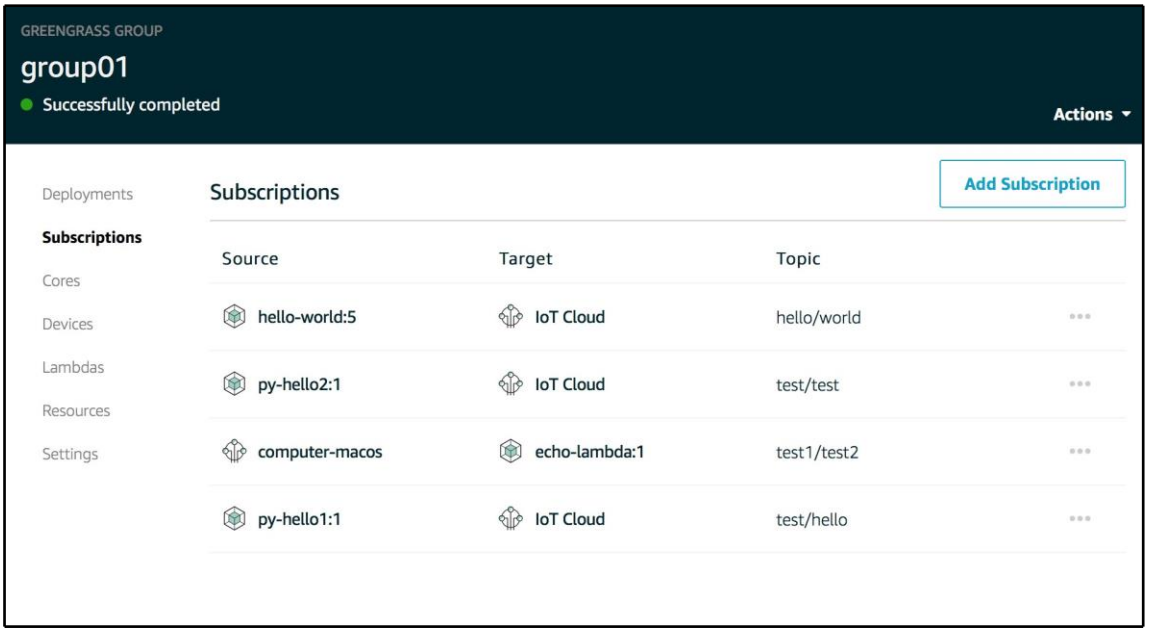
To deploy the AWS Lambda function to AWS Greengrass Core, you should publish the AWS Lambda function and then deploy it into AWS Greengrass Group:

1. From AWS Greengrass Group, you can add the existing Lambda function that we have already created. You just select your Lambda and its version and click on **Next**. Then, select **on-demand function** for the Lambda life cycle:

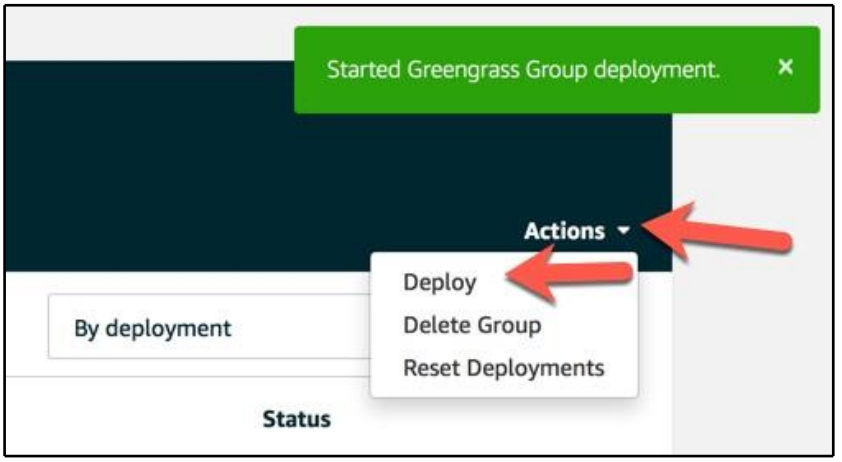


2. Furthermore, we define the subscription on AWS Greengrass. In this scenario, we add a thing, such as a computer, that will be used to invoke this Lambda. You can do it on AWS IoT Management Console.
3. You add a new subscription using the following configuration:
    - **Source**: Your thing or device
    - **Target**: Your Lambda function that you have created
    - **Topic**: test1/test2

You can see my subscriptions for the topic **test1/test2** in the following screenshot:



4. After completion, you can deploy this configuration into the AWS Greengrass Core machine, Raspberry Pi 3. Make sure the AWS Greengrass Core service has already started:



We have deployed AWS Greengrass Core. Next, we will test it by accessing from other IoT devices.

Accessing AWS Lambda from IoT devices

All API SDK for accessing AWS IoT, including AWS Greengrass, can be obtained on the software menu from AWS IoT Management Console. You can install it based on your device and runtime platforms.

In this section, I will show you how to invoke Lambda from AWS Greengrass Core. For testing, I used the Node.js application. You should install the `aws-sdk` library for Node.js. You also need to access the key ID and secret access key from the AWS IAM.

Please write the following complete program:

```
var AWS = require('aws-sdk');
AWS.config.update({region:'<region>'});

var accessKeyId = '<accessKeyId>'
var secretAccessKey = '<secretAccessKey>';
AWS.config.update({accessKeyId: accessKeyId, secretAccessKey:
secretAccessKey});

var lambda = new AWS.Lambda();
var params = {
 FunctionName: '<lambda-function-arn>',
 Payload: '{"msg": "this is testing"}'
};
lambda.invoke(params, function(err, data) {
 if (err) console.log(err, err.stack);
 else console.log(data);
});
```

Now you can run the preceding program using the following command:

```
$ node lambda_invoker.js
```

On successful execution, you should see a message on the Terminal that shows the status and response message from Lambda. A sample of the program output can be seen in the following screenshot:

```
agusk$ node lambda_invoker.js
{ StatusCode: 200,
  ExecutedVersion: '$LATEST',
  Payload: '{"msg": "this is testing"}' }
agusk$
```

We have accessed AWS Lambda from an IoT device or computer. You can do more experiments to get experience with AWS Lambda.

Next, we will integrate AWS Lambda with other AWS services.

Building IoT projects with AWS Lambda and AWS Greengrass

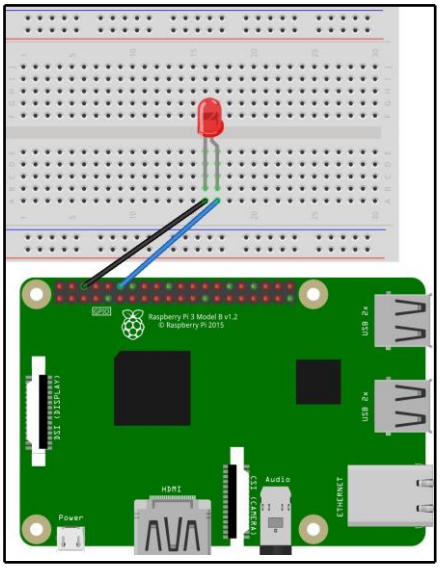
Integrating AWS Lambda and AWS Greengrass can leverage your IoT business. In this section, we will explore IoT projects to implement AWS Lambda and AWS Greengrass. We will build a simple app to monitor a local resource, such as a sensor.

Accessing local resources

AWS Greengrass can access local resources. This is possible if we use AWS Greengrass version 1.3.0 or later versions. You can configure an AWS Lambda to access specific resources. The following is a list of local resources that you can apply:

- Folders and files
- Serial ports
- USB
- GPIOs
- GPUs
- Cameras

To show you how to work with local resources in the AWS Greengrass Core machine, we will access local GPIO resources. We need an LED that is attached to Raspberry Pi on GPIO 18 (pin 12). You can see this wiring in the following image:



Our aim is to develop a blinking Lambda. This demo will make a blinking LED while the Lambda function is called. The next step is to develop AWS Lambda and then configure the AWS Greengrass Core machine.

Developing local AWS Lambda

We use Node.js to develop local AWS Lambda. Firstly, we write the Node.js program with the Raspberry Pi environment. Open the Raspberry Pi Terminal and create a folder called `blinking`. We will use the `node-rpio` library to access the Raspberry Pi GPIO. This library can be found at `https://github.com/jperkin/node-rpio`.

1. Navigate to the `blinking` folder from the Terminal. Then, we install the `node-rpio` library by typing the following command:

```
$ npm install rpio
```

2. You also need to configure AWS Lambda to enable working with `gpiomem`. You can do that by typing the following command:

```
$ sudo cat >/etc/udev/rules.d/20-gpiomem.rules <<EOF
SUBSYSTEM=="bcm2835-gpiomem", KERNEL=="gpiomem", GROUP="gpio",
MODE="0660"
EOF
```

3. Now, create the `index.js` file to write our blinking program. You can write this complete program for the `index.js` file as follows:

```
const ggSdk = require('aws-greengrass-core-sdk');
var rpio = require('rpio');

const iotClient = new ggSdk.IotData();
const os = require('os');
const util = require('util');

function publishCallback(err, data) {
 console.log(err);
 console.log(data);
}

...

setTimeout(greengrassBlinkingdRun, 2000);

exports.handler = function handler(event, context) {
 console.log(event);
 console.log(context);
};
```

4. Compress the `index.js` file included in the `node_modules` folder into a file called `blinking.zip`. Now you can upload the `blinking.zip` file into AWS Lambda Management Console.

Since this Lambda function accesses local resources, this Lambda cannot perform Lambda testing on AWS Lambda Management Console. You should only perform publishing directly.

Explanation: The following program will call the `greengrassBlinkingRun()` function once. Inside the function, it will open GPIO 18 (12 pin) and write the ;`HIGH/LOW` value into the GPIO. The program will also publish a message using the `publish()` function:

```
function greengrassBlinkingdRun() {
 rpio.open(12, rpio.OUTPUT, rpio.LOW);
 rpio.write(12, rpio.HIGH);
 iotClient.publish(pubOpt, publishCallback);
 rpio.sleep(1);

rpio.write(12, rpio.LOW);
 state = rpio.LOW;
 iotClient.publish(pubOpt, publishCallback);
 rpio.sleep(1);
```

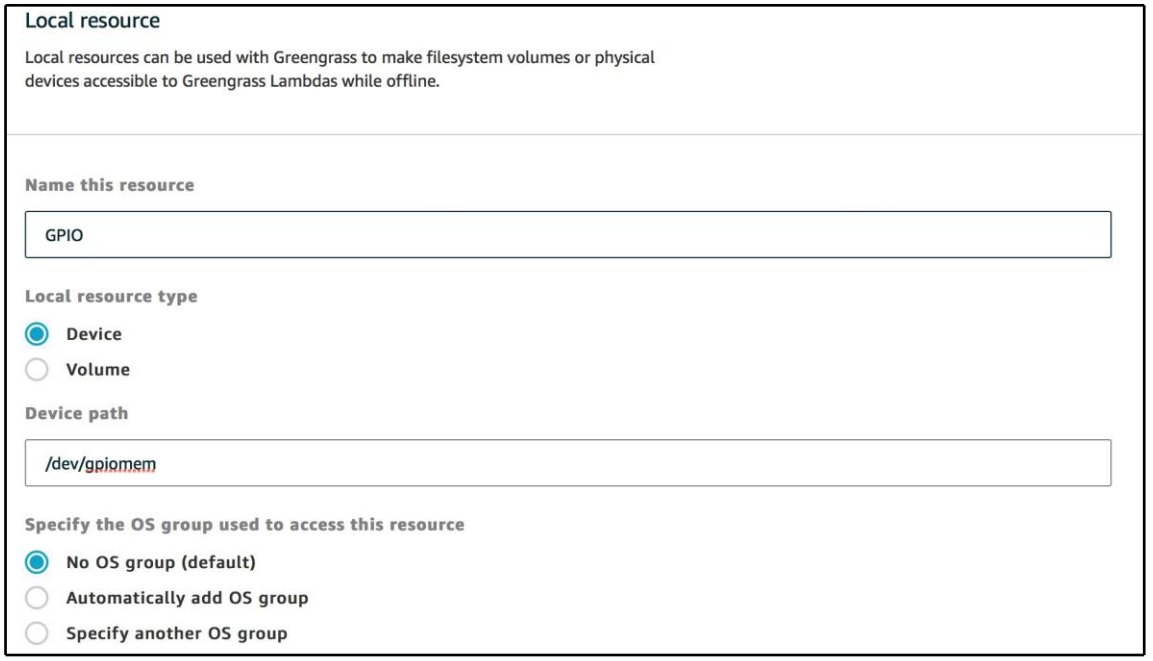
```
 rpio.close(12);
}
setTimeout(greengrassBlinkingdRun, 2000);
```

Configuring AWS Greengrass Core

After we have published AWS Lambda, we can consume this Lambda function into AWS Greengrass. Firstly, we need to prepare our resources for the local AWS Greengrass Core.

Follow these steps to configure local resources on AWS Greengrass Core:

1. Open AWS Greengrass Group and then add new local resources from the **Resources** menu. Fill in the resource name; for instance, `GPIO`. Select the **Device** option for **Local resource type** with the `/dev/gpiomem` path, as shown in the following screenshot:

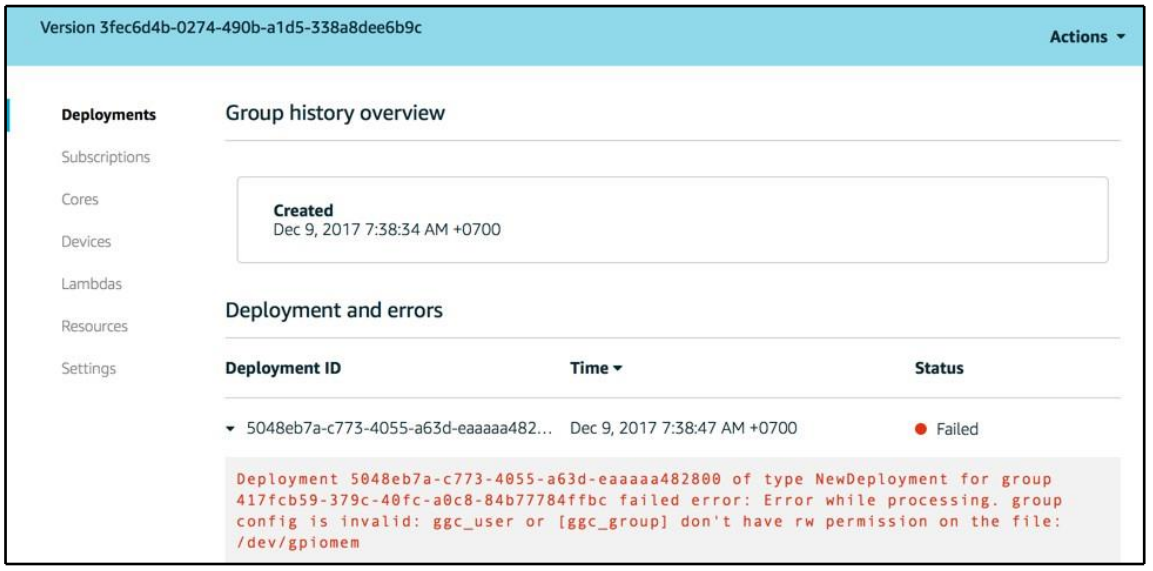


Save this resource.

2. Now, create Lambda on AWS Greengrass Group from the existing Lambda that we have already created. After this Lambda is created, you can add our GPIO resource. You can enable the **Read access to /sys directory** option to ensure our Lambda can access local resources.
3. You should also add a new subscription for our local Lambda. Set the source from our local Lambda and target for the IoT Cloud. You can set any topic, for instance, pi/blinking.
4. Before deploying to AWS Greengrass Core, you should configure your Raspberry Pi 3. Regarding the security issue, you should configure `ggc_user` as a member of the GPIO group. Please execute the following command on the Raspberry Pi Terminal in order to configure `ggc_user` as the member of `gpio`:

```
$ sudo adduser ggc_user gpio
```

If not configured, you will probably get problems, as shown in the following screenshot:



5. Reboot your Raspberry Pi 3 and then start the Greengrass service after the reboot is completed.

Now you can deploy AWS Greengrass into the AWS Greengrass Core machine.

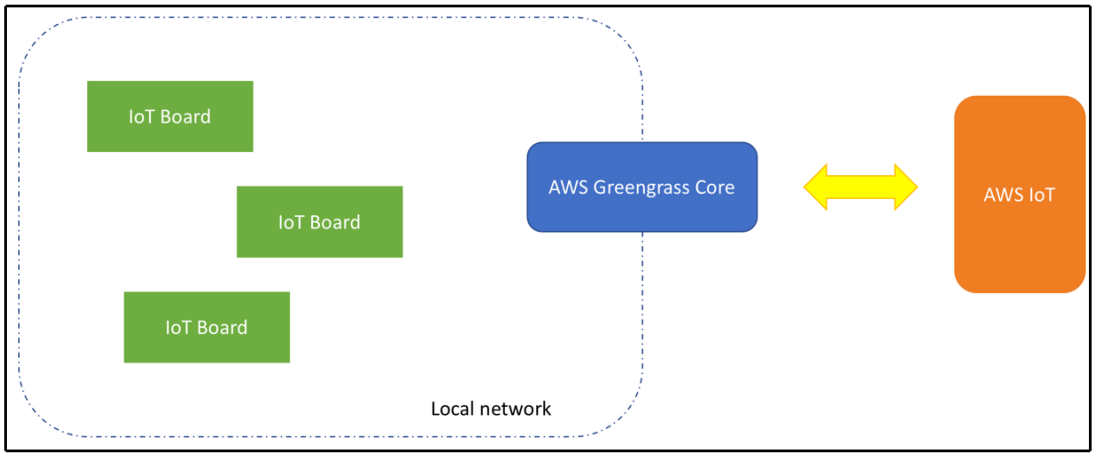
Testing the demo

You can run this demo using the **Test** website from AWS IoT Management Console. You can set it to subscribe with `pi/blinking`. You should see the blinking LED once.

Interacting with things within a group

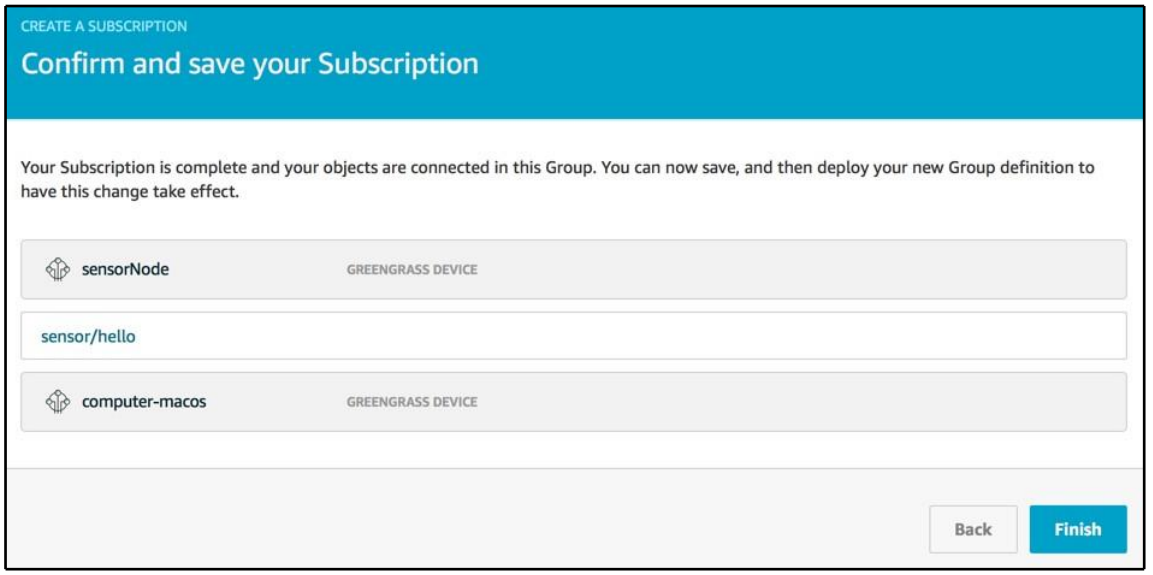
In this section, we will develop an application to interact between IoT devices within the AWS Greengrass Group. This feature enables our IoT devices to consume resources from other IoT devices. To work with this case, we should register all IoT devices in the AWS IoT Management Console. Then, add those IoT devices into AWS Greengrass Group.

We can access an IoT board within one group from AWS Greengrass Core. AWS IoT can interact with IoT boards through AWS Greengrass Core. You can see this in the following figure:



For a demo, we will develop publisher and subscriber applications to show how those IoT devices communicate with each other. For testing, we use a sample program from AWS IoT Python SDK at `https://github.com/aws/aws-iot-device-sdk-python`.

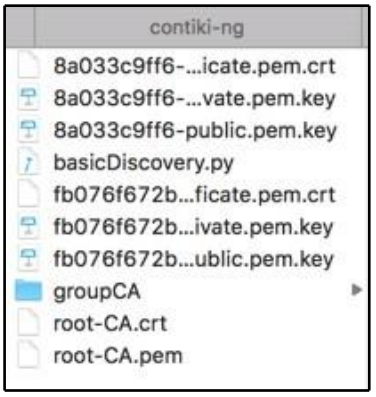
1. Firstly, add at least two IoT devices into the current AWS Greengrass Group. Then, on the **Subscriptions** menu, add a new subscription.
2. Select the source for any IoT device name and target for the other IoT device name. Set the subscription topic, for instance, `sensor/hello`. You can see this in the following screenshot:



3. Download all the certificates and private key files from your IoT devices and save them into one folder. You should install AWS IoT Python SDK on your IoT devices. In this case, it is my computer. You can install this SDK using the following command:

```
$ pip install AWSIoTPythonSDK
```

4. Now you can download a sample program from `https://github.com/aws/aws-iot-device-sdk-python/blob/master/samples/greengrass/basicDiscovery.py`. Then, save it into one folder with our IoT device certificate and key files:



5. Now you can open the Terminal and run the publisher program as follows:

```
python basicDiscovery.py -e host_aws_greengrass_core -r root-
CA.pem -c iotdevice1-certificate.pem.crt -k iotdevice1-
private.pem.key -n iotdevice1-name -m publish -t 'sensor/hello'
-M 'This message from sensorNode'
```

Here:

- `host_aws_greengrass_core` is the hostname or name from the AWS Greengrass Core machine
- `root-CA.pem` is the certificate file for AWS
- `iotdevice1-certificate.pem.crt` is a certificate file from IoT device 1
- `iotdevice1-private.pem.key` is a private key from IoT device 1
- `iotdevice1-name` is the IoT device 1 name
- `sensor/hello` is the subscription topic
- `This message from sensorNode` is a message sample

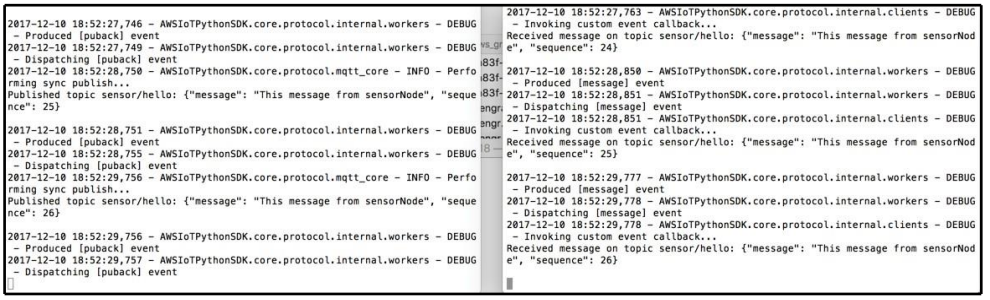
6. Furthermore, we run a program for the subscriber application as follows:

```
python basicDiscovery.py -e host_aws_greengrass_core -r root-
CA.pem -c iotdevice2-certificate.pem.crt -k iotdevice2-
private.pem.key -n computer-macos -m subscribe -t
'sensor/hello'
```

Here:

- `host_aws_greengrass_core` is the hostname or name from the AWS Greengrass Core machine
- `root-CA.pem` is the certificate file for AWS
- `iotdevice2-certificate.pem.crt` is a certificate file from IoT device 2
- `iotdevice2-private.pem.key` is a private key from IoT device 2
- `iotdevice2-name` is the IoT device 2 name
- `'sensor/hello'` is the subscription topic

On successful execution, you should see a message that is sent and received by the programs, as shown in the following screenshot:



Summary

We have learned how to work with local Lambda on AWS Greengrass. We also developed programs to access local resources and communicate between IoT devices.

In the next chapter, we will learn how to work AWS IoT Button and apply it in your IoT projects.

Reference

Kurniawan, Agus. 2018. Learning AWS IoT. Packtpub. https://www.packtpub.com/virtualization-and-cloud/learning-aws-iot