SSDIOT Workshop

Building IoT solution with the cloud

Darryl Ng Eng Soon

Institute of

Systems Science

Singapore 119 615

darryl.ng@nus.edu.sg

(65) 6516 2092

Workshop 1: Basic Linux command

1. Check the release version of raspbian.
   cat /etc/os-release

2. Check the linux standard base version of the operating system.
   lsb_release -a

3. Check the kernel version of your linux operating system in RaspberryPI.
   uname -a

4. Check if you have enough space on your SD card.
   df -h

5. Update pre-installed linux packages using the package manager apt.
   sudo apt-get update

6. Check the ip address of your Raspberry PI.
   ifconfig

7. Check if pip is installed.
   pip --version

   The below steps are for your info. Raspberry PI image comes with Python and pip installed.

   If you don't have pip, install pip with the script provided by Python Packaging Authority.
   a. Download the installation script.
      curl -O https://bootstrap.pypa.io/get-pip.py

   b. Run the script with Python.
      python get-pip.py --user

8. Check if python version 2 and/or version 3 is/are installed.
   a. **python --version**
   b. **python3 --version**

9. Check where Python is installed.
   a. **which python**

10. Check your OS path and available Shell.
    a. **echo $PATH**
    b. **echo $SHELL**

Workshop 2: Setting Up Raspberry PI for subsequent workshops

This workshop is for your reference.
(You may follow through this workshop and execute the steps required to setup a new Raspberry PI to connect to AWS cloud services through command line interface.)

For the purpose of subsequent workshops, the Raspberry PI has been setup based on the steps listed below.

Install AWS CLI

1. Use pip to install the AWS CLI.
   pip install awscli --upgrade --user

2. Add the AWS path to your Raspberry PI profile.
   a. **sudo nano ~/.profile**
   b. **PATH = $~/.local/bin:$PATH**

3. Reboot the Raspberry PI.
   sudo reboot

4. Verify AWS CLI was installed correctly.
   aws --version

Install GrovePi (Do not install the Grovepi shield yet)

1. **sudo apt-get update**
2. **sudo apt-get install emacs –y**
3. **cd /home/pi/Desktop**
4. **sudo git clone https://github.com/DexterInd/GrovePi**
5. **cd /home/pi/Desktop/GrovePi/Script**
6. **sudo chmod +x install.sh**
7. **sudo ./install.sh**
8. **sudo reboot**
9. **sudo pip install grovepi**
10. **sudo shutdown –h now**

Check GrovePi firmware

1. **cd Desktop/GrovePi/Software/Python**
2. **sudo python grove_firmware_version_check.py**

Update downloaded GrovePi repo

1. **cd /home/pi/Desktop/GrovePi**
2. **sudo git fetch origin**
3. **sudo git reset --hard**
4. **sudo git merge origin/master**

Update GrovePi Firmware

1. Go to the Firmware folder in the GrovePi folder.
   cd /home/pi/Desktop/GrovePi/Firmware

2. Make the firmware_update.sh bash script executable.
   sudo chmod +x firmware_update.sh

3. Run the update script to check and proceed with the firmware update.
   sudo ./firmware_update.sh

Install AWS IoT Device SDK for Python

1. Navigate to Documents directory.
   cd /home/pi/Documents

2. Create a project folder.
   mkdir groveproject

3. Navigate to the project directory.
   cd /home/pi/Documents/groveproject

4. Build from source.
   a. **git clone https://github.com/aws/aws-iot-device-sdk-python.git**
   b. **cd aws-iot-device-sdk-python**
   c. **python setup.py install**

Revert to a tested working Raspbian kernel for GrovePi (optional)

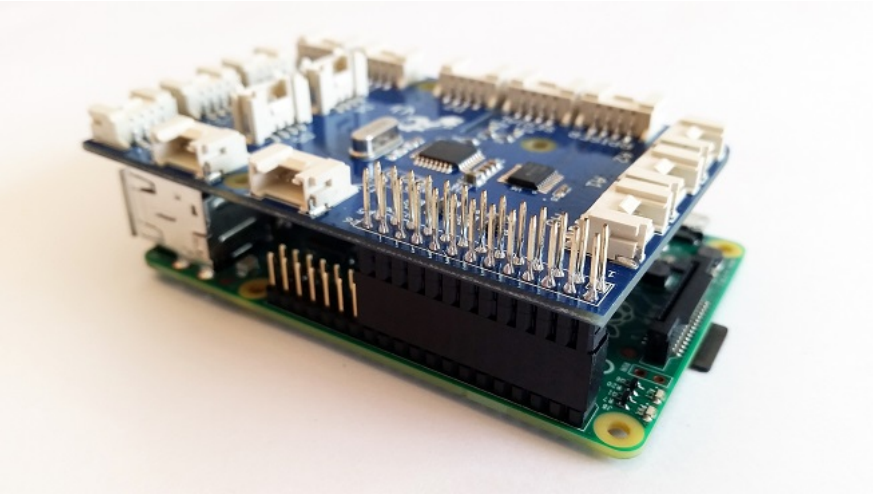
1. Check the kernel version.
   uname –a

2. Downgrade the kernel to version 4.4.50-v7+.
   sudo rpi -update 52241088c1da59a359110d49c1875cda56496764

Workshop 3: Assemble the GrovePI shield and sensors on Raspberry PI

1. Connect the GrovePi to Raspberry PI.
   a. Mount your GrovePi (blue board) on the Raspberry Pi (green board)

The GrovePi has a black plastic piece on the bottom, which fits perfectly with the metal pins sticking out of the Raspberry Pi. Slide the GrovePi board onto the pins on the Raspberry Pi as shown in the pictures below.



Ensure that the pins are properly aligned when stacking the GrovePi, and push down until they go in all the way.

2. Check and see if the Raspberry Pi is able to detect the GrovePi.
   sudo i2cdetect -y 1

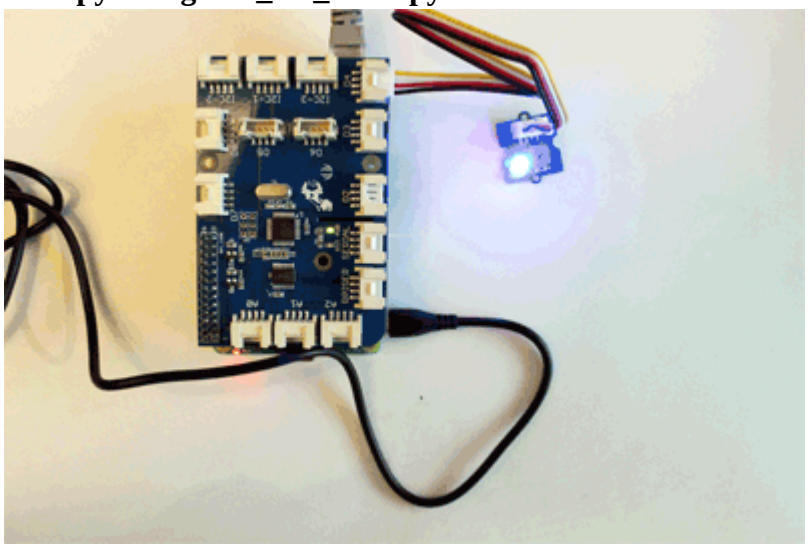
```
pi@raspberrypi ~/Desktop/GrovePi/Firmware $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- 04 -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi ~/Desktop/GrovePi/Firmware $
```

If you can see a "**04**" in the output, this means the Raspberry Pi is able to detect the GrovePi.

3. Connect a Grove LED to GrovePi port labeled D4.

4. Go to the Python folder in the GrovePi repository.
   cd Desktop/GrovePi/Software/Python

5. Blink the LED.
   sudo python grove_led_blink.py



6. Type "ctrl + c" to stop the program from running.

7. Attach Grove Sensors Before Startup.
   a. Connect the **DHT sensor to digital port 7**.
   b. Connect the **RGB LCD display** to any of the **I2C ports**.
   c. Connect the **Grove LED** to **digital port D4**.
   d. Connect the **Grove Light Sensor** to **analog port A0**.
   e. Connect the **Grove Button** to **digital port D3**.
   f. Connect the **Grove Relay** to **digital port D4**.

Note:

It is good practice to attach the Grove Sensors that you want to use before powering on the RaspberryPi. The Raspberry Pi may restart if you attach a sensor when the Raspberry Pi is powered.

Workshop 4: Programming the sensors

This workshop is requires you to have some programming background in Python language.
(You may refer to details on the following URL https://www.iss.nus.edu.sg/executive-education/course/detail/working-with-data-cloudops-and-things-using-python/startup-and-sme to know more on our Python course.)

In this workshop, you will write python code in a python file named **ssdiot01.py** in the project folder **groveproject**.

1. Use the import statement to include the modules required to support the running of this piece of code to take with sensors.

```
from grove_rgb_lcd import *
from time import sleep
from math import isnan
import grovepi
```

2. Define the port number and properties for DHT sensor.

```
# connect the DHT sensor to digital port 7
dht_sensor_port = 7

# 0 for blue-colored sensor and 1 for white-colored sensor
dht_sensor_type = 0
```

3. Define the port number and threshold properties for Light sensor.

```
# Connect the Grove Light Sensor to analog port A0
light_sensor = 0


# Turn on LED once sensor exceeds threshold resistance
threshold = 10
```

4. Define the port number for LED.

```
# Connect the LED to digital port D4
led = 4
```

5. Define the port number for Relay.

```
# Connect the Grove Relay to digital port D3
relay = 3
```

6. Write a function named **initialFunc()** to initialize the sensors' input and output.

```
def initialFunc():
        # set green as backlight color
        # we need to do it just once
        # setting the backlight color
        # to reduce the amount of data transfer over the I2C line
        setRGB(0,255,0)

        grovepi.pinMode(light_sensor,"INPUT")
        grovepi.pinMode(led,"OUTPUT")

        grovepi.pinMode(pir_sensor,"INPUT")

        return
```

7. Write a function named **readValues()** to read the value from different sensors.

```
def readValues():

        try:
                # Get light sensor value
                light = grovepi.analogRead(light_sensor)
                print("light sensor_value = ",light)
        except IOError as TypeError:
                print ("Error")
                return [-1, -1, -1]

        sleep(0.2)

        try:
                # Get temperature and humidity from DHT sensor
                [ temp,hum ] = grovepi.dht(dht_sensor_port,
dht_sensor_type)
                print("temp =", temp, "C\thumidity =", hum,"%")
        except (IOError, TypeError) as e:
                print(str(e))
                return [-1, -1, -1]

        return [light, temp, hum]
```

8. Write a function named **processValue(l1)** to process the value retrieved from light sensor.

```
def processValue(l1):
        # Calculate resistance of sensor in K
        resistance = (float)(1023 - l1) * 10 / l1
        print("light sensor_value = %d resistance = %.2f" %(l1,
resistance))

        return resistance
```

9. Write a function named **triggerLight(resistance)** to turn on LED when the light sensor sense that the light density is low.

```
def triggerLight(resistance):
        if resistance > threshold:
             # Send HIGH to switch on LED
             grovepi.digitalWrite(led,1)
         else:
             # Send LOW to switch off LED
             grovepi.digitalWrite(led,0)
        return
```

10. Write a function named **triggerLCD(temp, hum)** to update the LCD with the values read from the DHT sensor.

```
def triggerLCD(temp, hum):
        t = str(temp)
        h = str(hum)

        # instead of inserting a bunch of whitespace,
        # we can just insert a \n
        # we're ensuring that if we get some strange strings
        # on one line, the 2nd one won't be affected
        setText_norefresh("Temp:" + t + "C\n" + "Humidity :" + h
+ "%")
        return
```

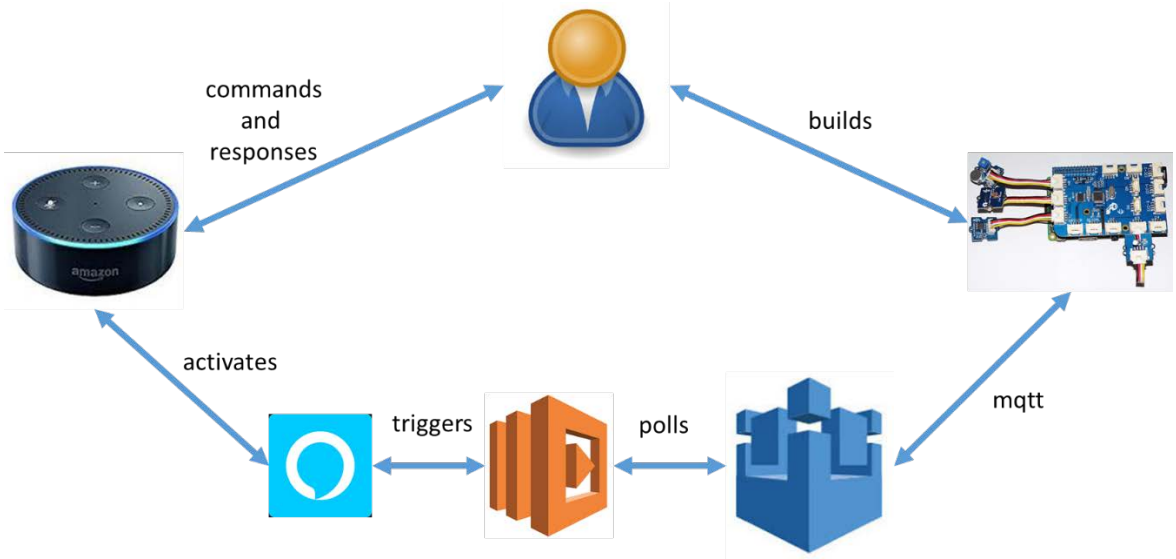
11. Complete the code by calling and executing the functions to read and process values from the sensors.

```
initialFunc()

while True:
        try:
                [ lightval, tempval, humval ] = readValues()

                resistance = processValue(lightval)
                triggerLight(resistance)
                triggerLCD(tempval, humval)

        except (IOError, TypeError) as e:
                print(str(e))
                # and since we got a type error
                # then reset the LCD's text
                setText("")

        # wait some time before re-updating the LCD
        sleep(2)
```

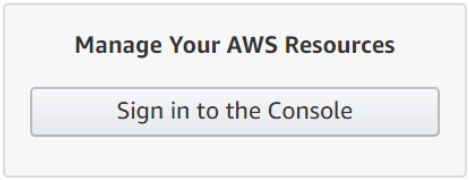
Workshop 5: Setup the AWS account and IoT Core service

In this workshop, we will show you how to work with AWS IoT. You will learn how to connect your Raspberry Pi with an attached sensor to AWS IoT and send measurement data into AWS. By the end of the next few workshops, you will be able to build an end-to-end IoT solution that you can interact with via voice interaction using Amazon Echo Dot to receive measurement data from your Raspberry Pi IoT thing with attached sensor.

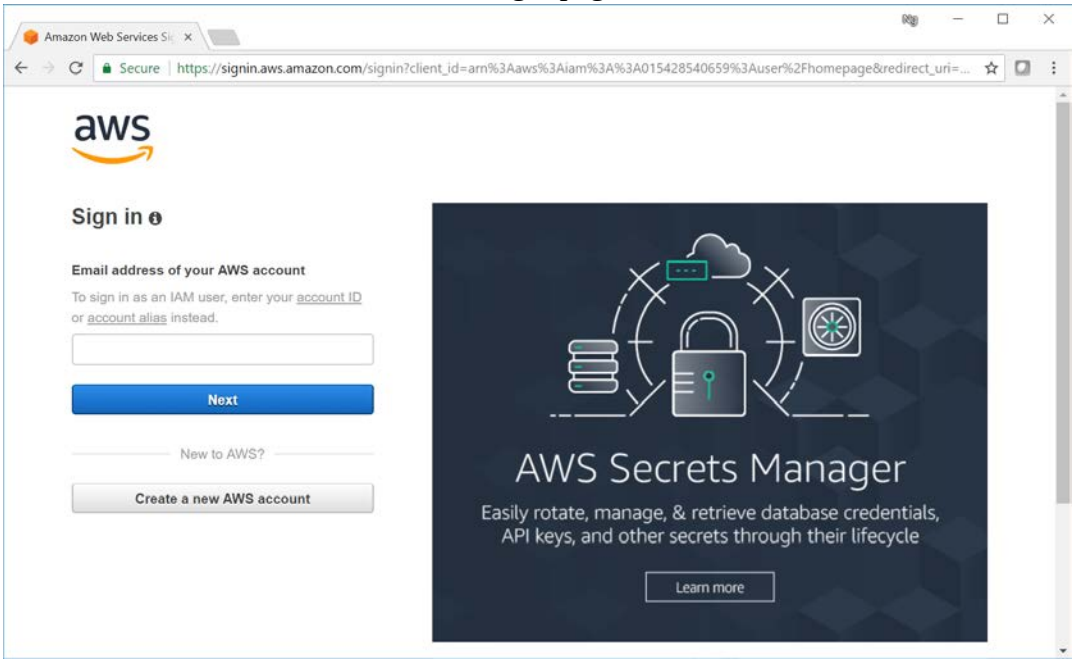
The end-to-end architecture will look like the following:



1. Navigate to AWS Management Console using the URL https://aws.amazon.com/console/. Click '**Sign in to the Console**' button as shown.

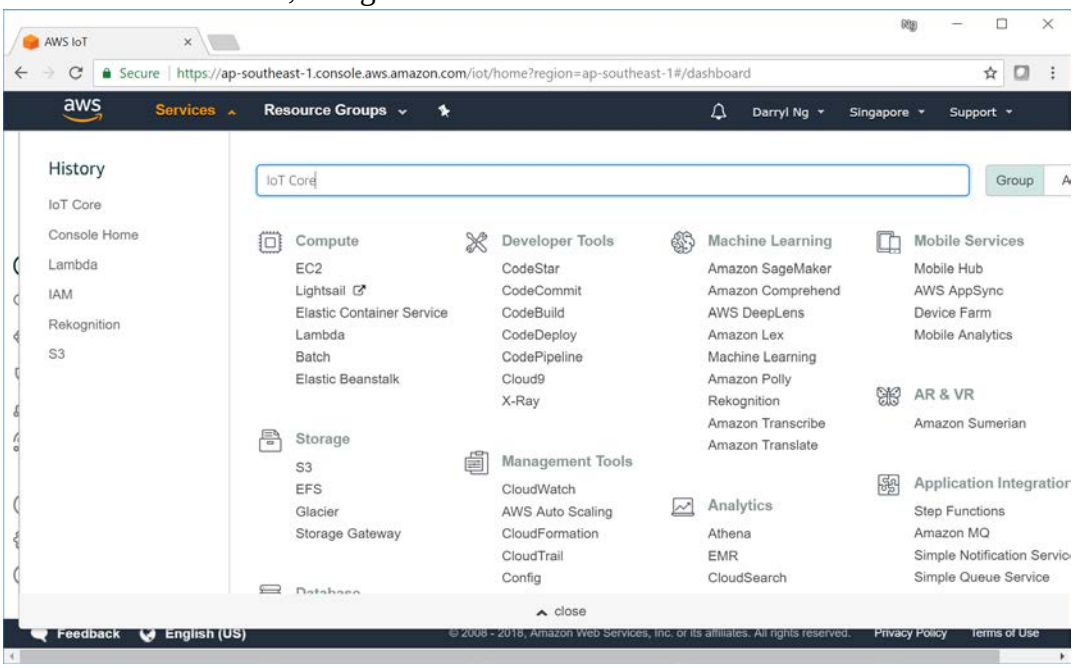


2. You will be redirect to AWS console login page as shown.

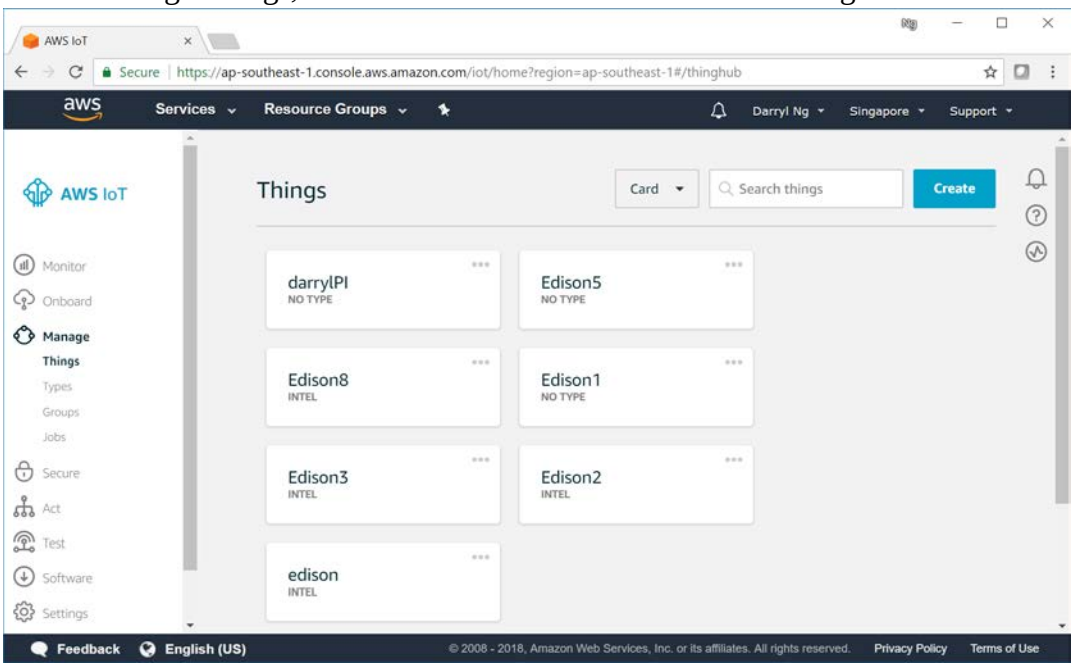


3. If you already have an AWS account, then sign in to the account with your AWS account ID and password. Otherwise, click '**Create a new AWS account**' button and follow through the steps and process to create a new AWS account. (We will not through the steps of creating a new AWS account here. Please seek your instructor's assistance where necessary.)

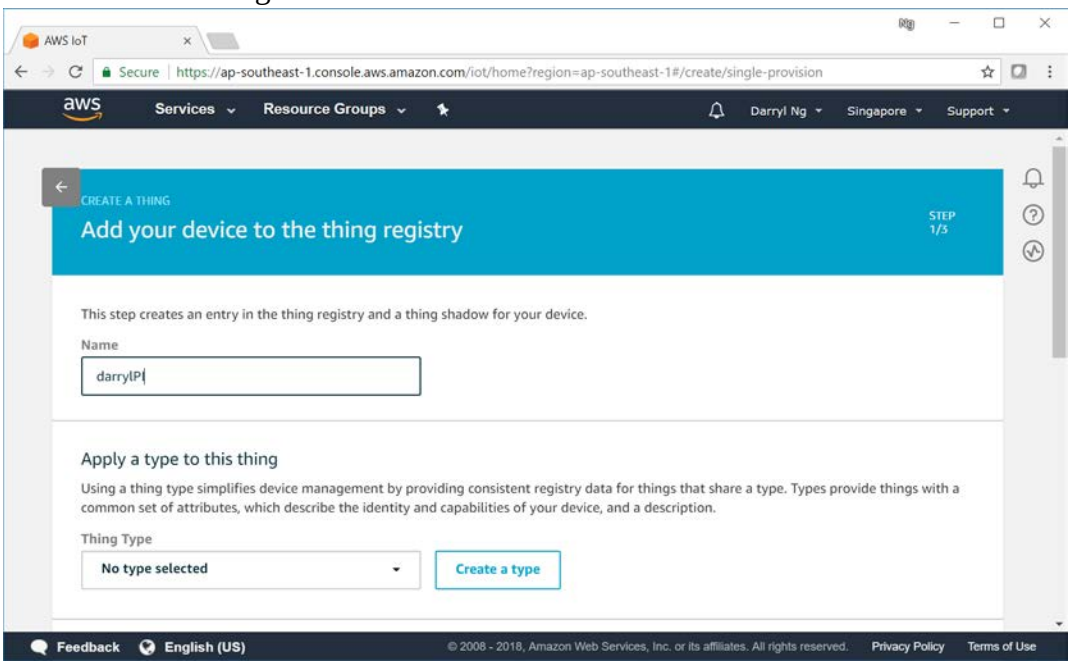
4. One the AWS console, navigate to **AWS IoT Core** Service.



5. Under Manage Things, click '**Create**' button to create a new thing.

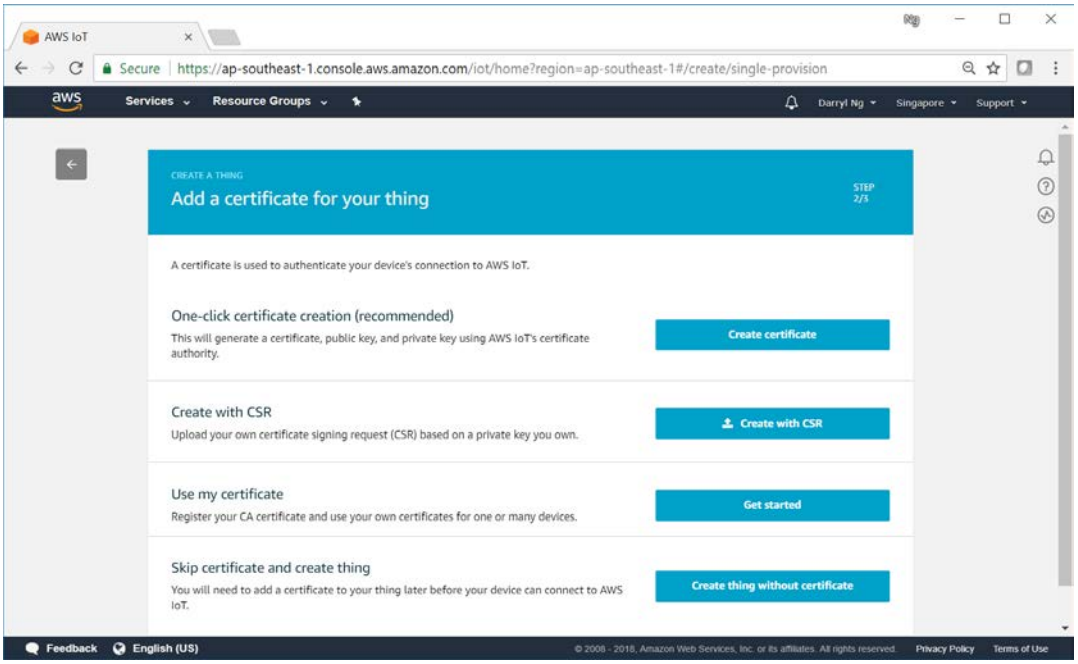


6. Provide a name that will correspond to the thing (Raspberry PI) you are going to add to the Device Management.

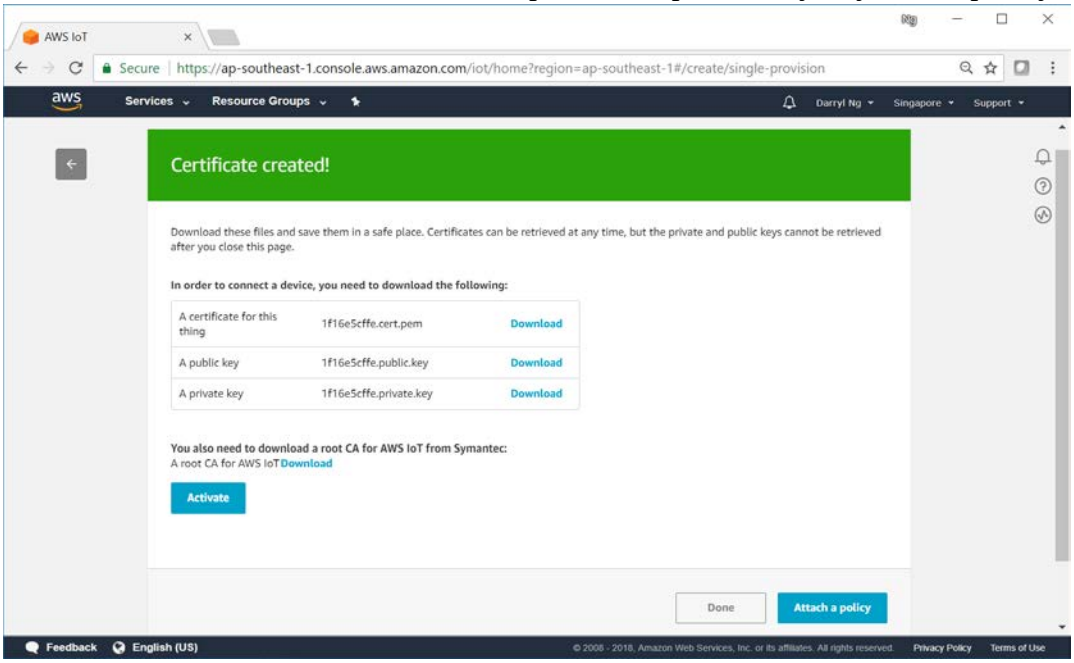


Click '**Next**' button at the bottom of the page to continue.

7. Click '**Create Certificate**' button for a one-click certificate creation.

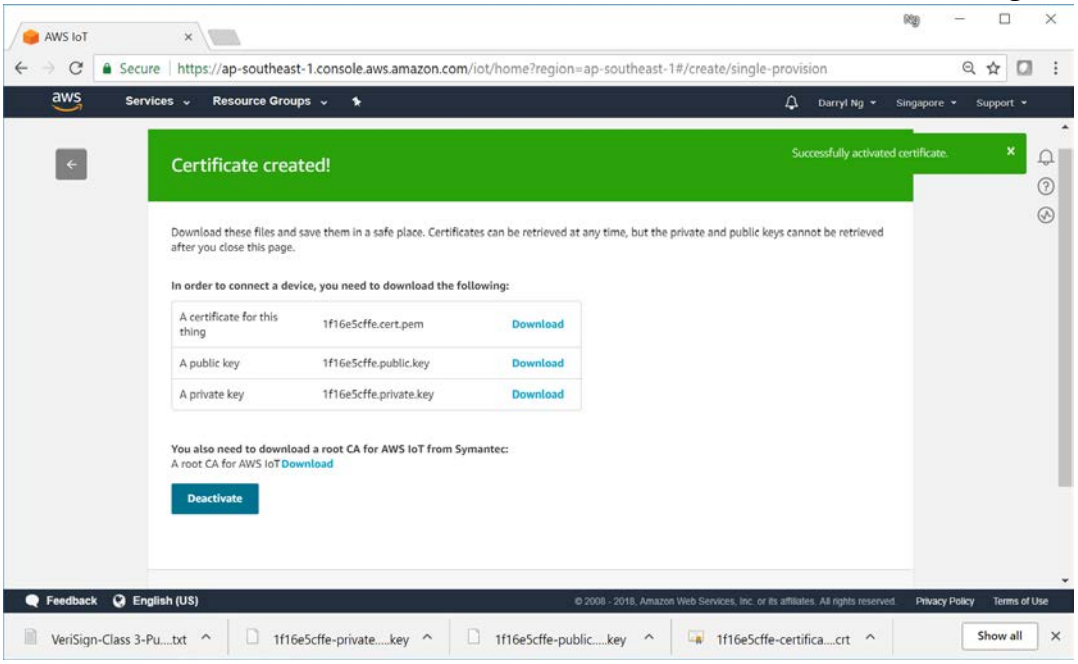


8. Download the certificate created and the public and private key to your Raspberry Pi.

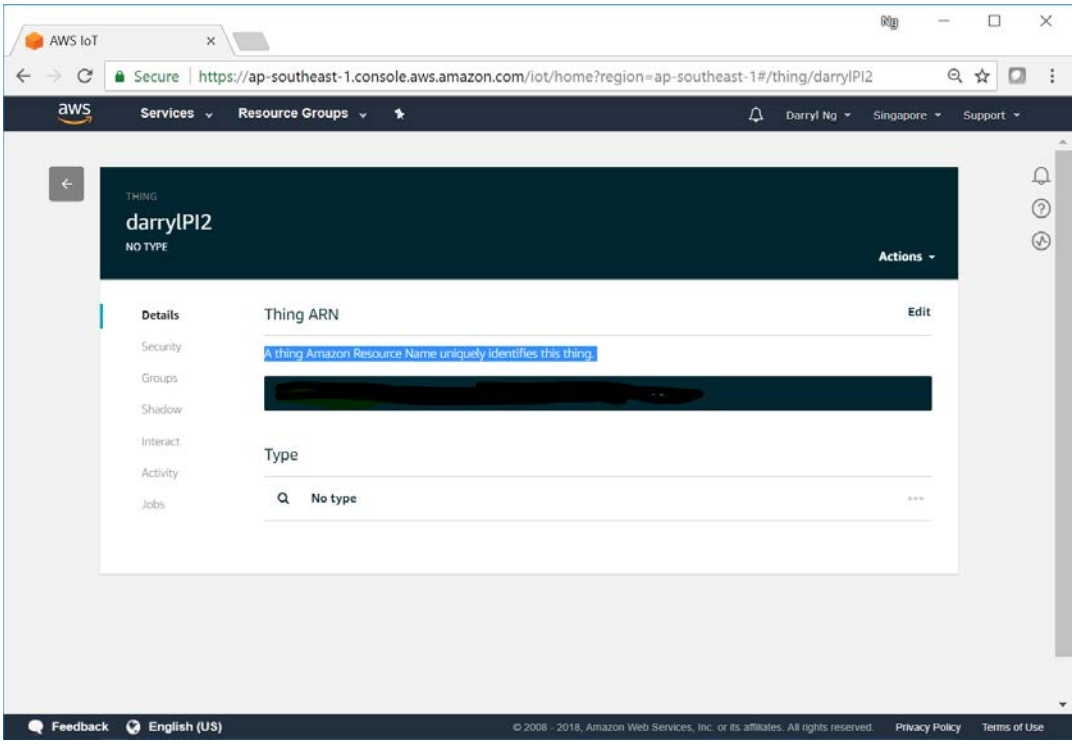


Also download the root CA for AWS IoT to your Raspberry Pi.

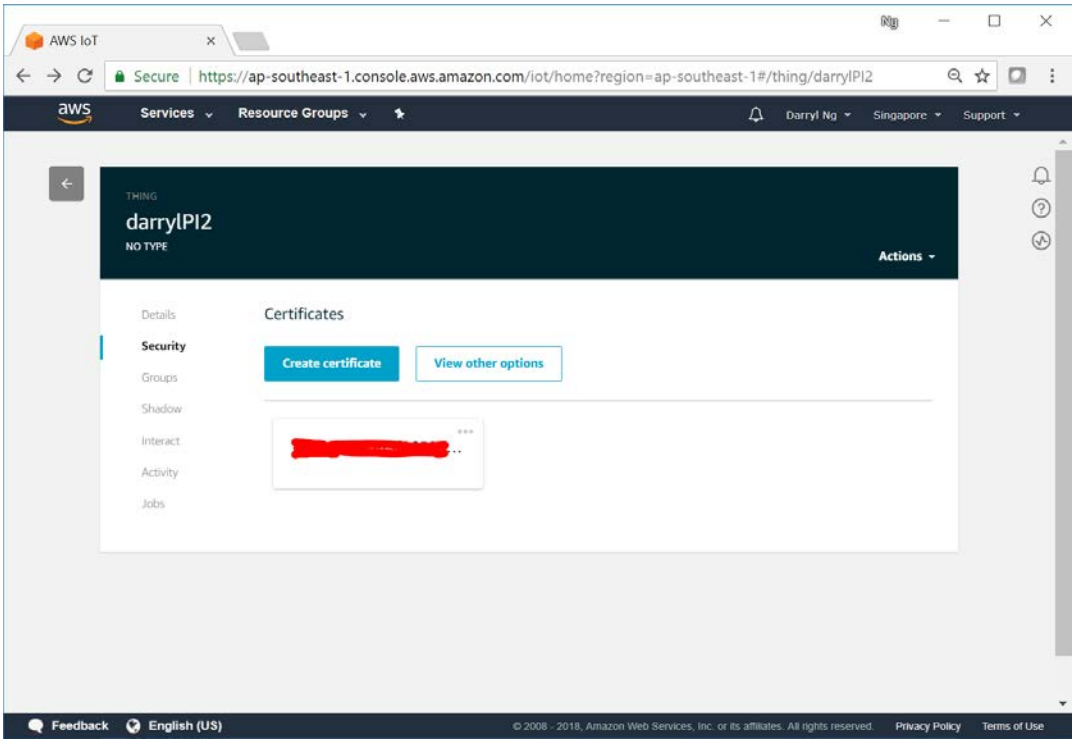
Last but not least, click '**Activate**' button to activate the certificate for the thing.



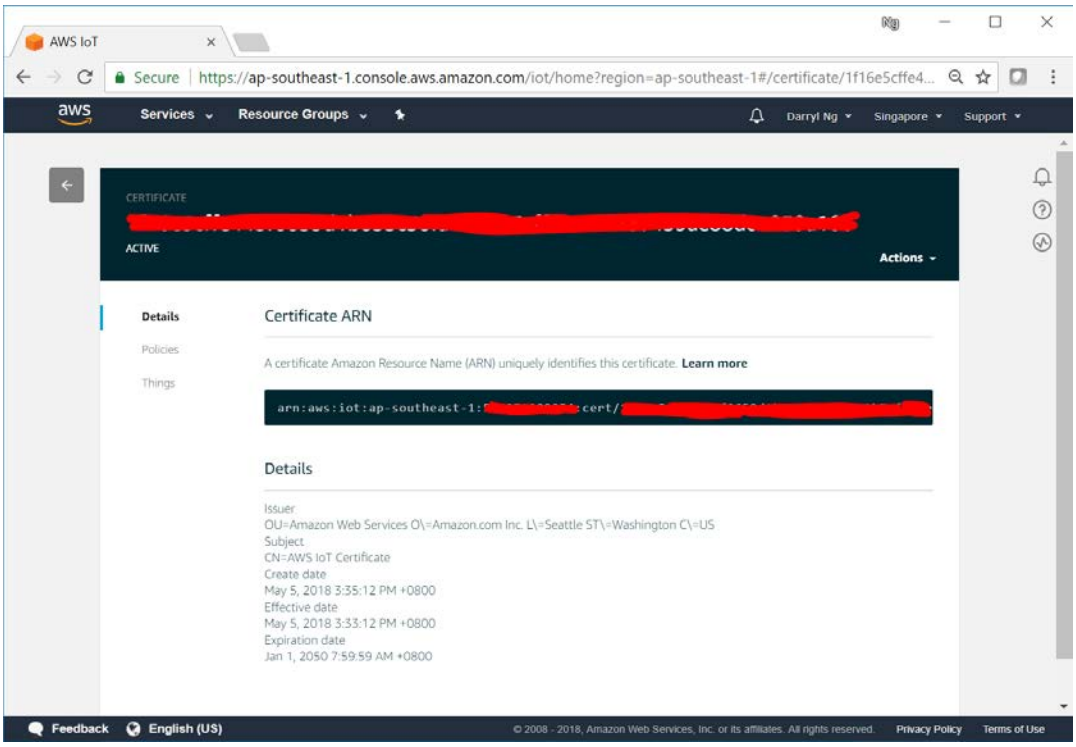
9. Under **Manage Things**, select the thing you have just created. You will be directed to a screen similar to the one shown.



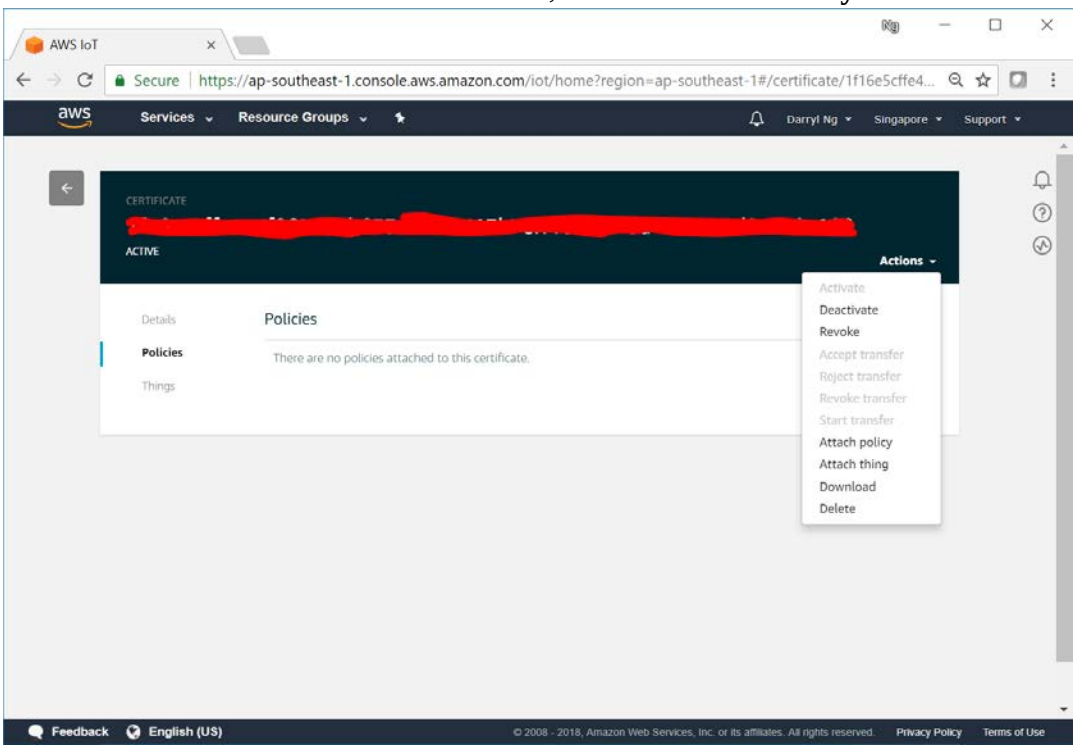
10. Click '**Security**'.



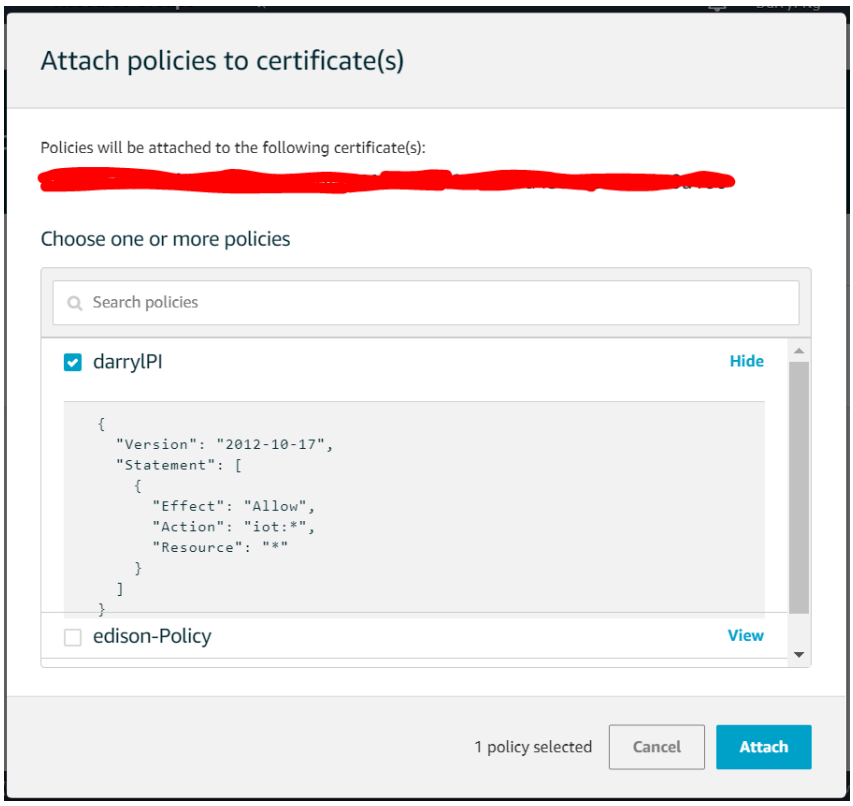
11. Click on the Created Certificate. You will see screen similar to the one shown.



12. Click '**Policies**'. From the '**Actions**' menu, choose '**Attach Policy**'.



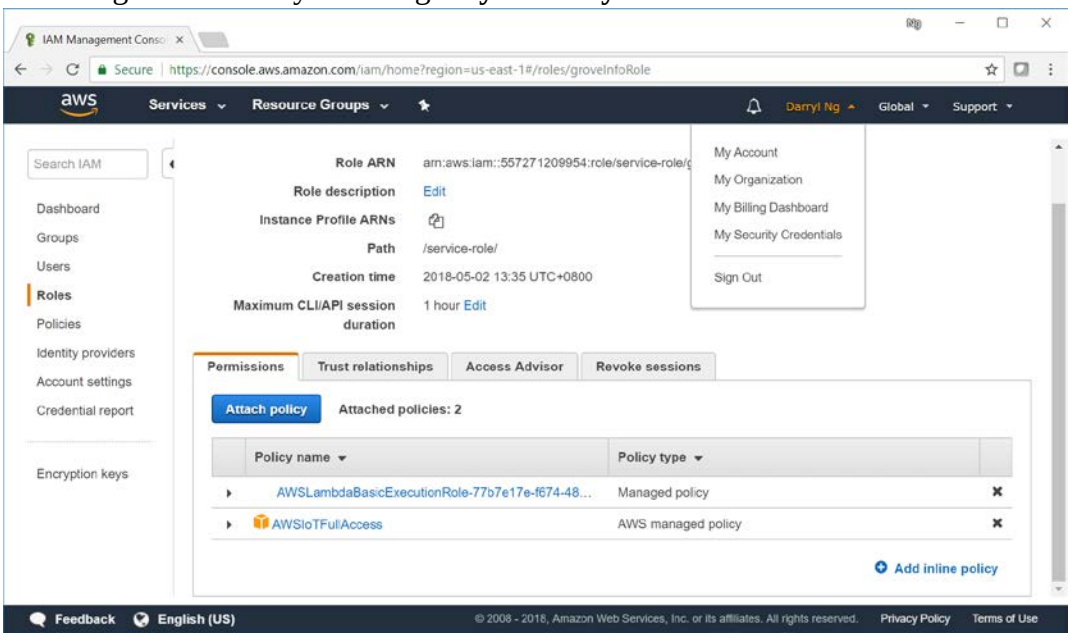
13. Select a policy and attach to the certificate that is attached to the thing you have created.



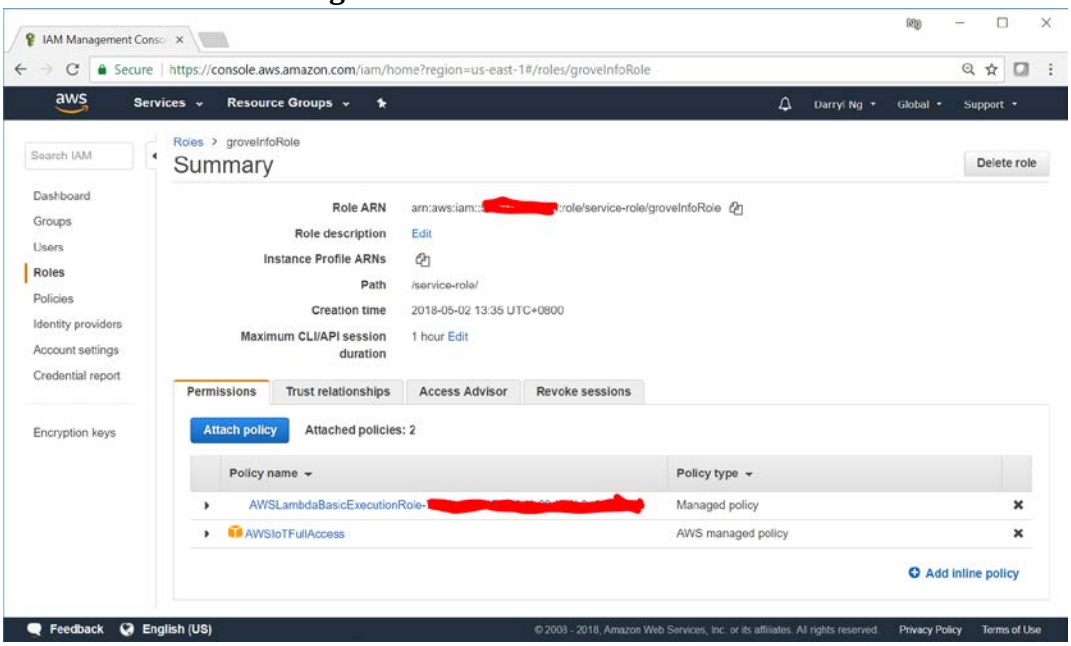
If you have no certificate for the AWS IoT service, your instructor will show and teach you how to create one.

Workshop 6: AWS Lambda Service

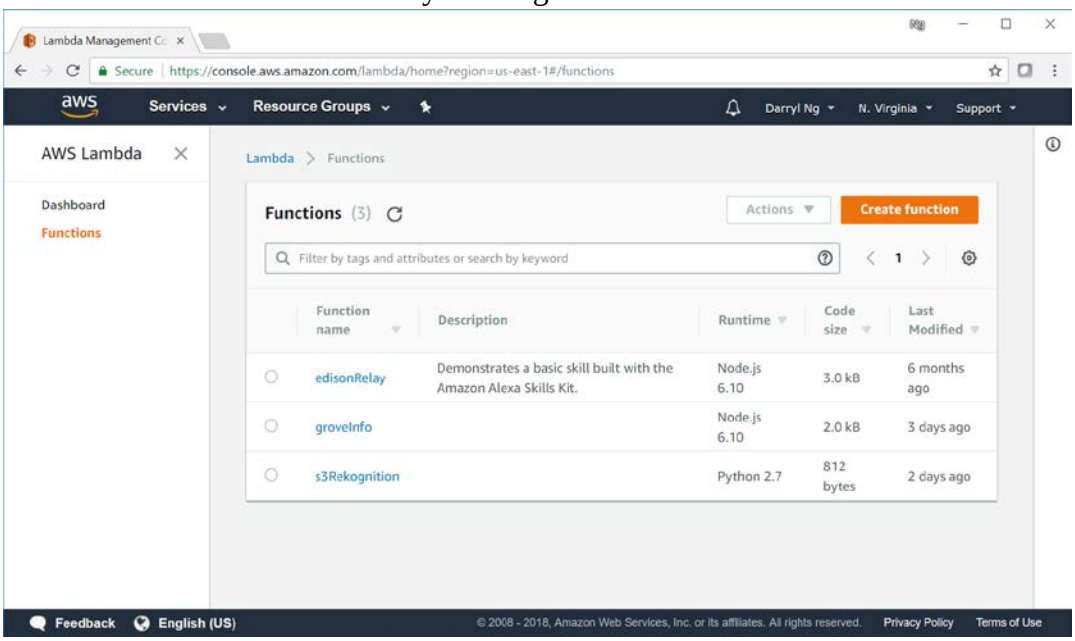
1. Before creating a Lambda function, we will create a role that will support the Lambda function to talk to the various AWS services. Click on the account dropdown menu and navigate to IAM by selecting '**My Security Credentials**'.



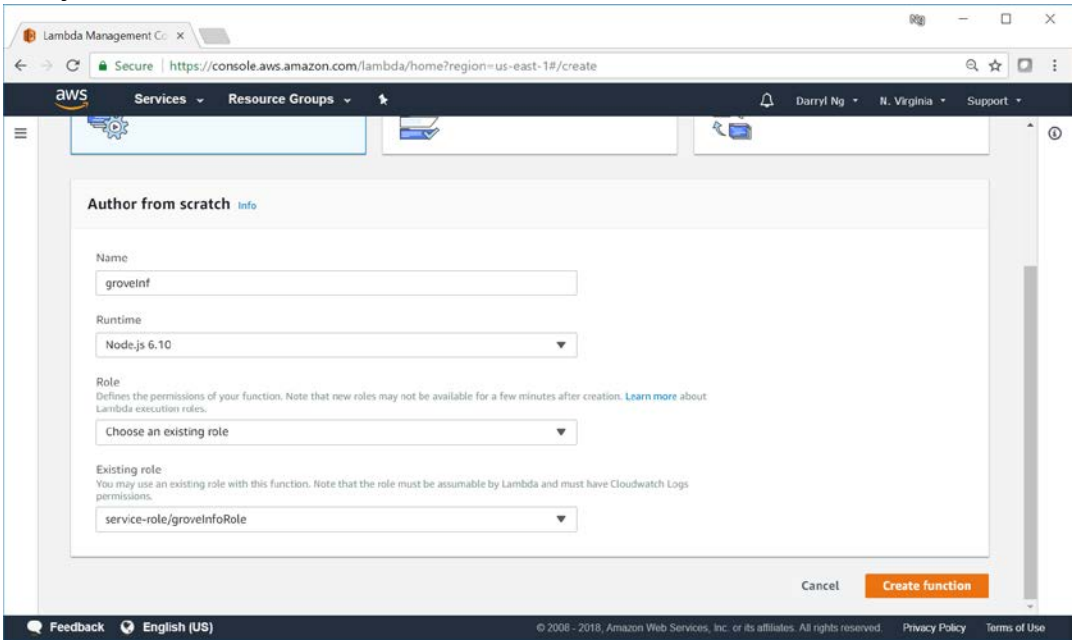
2. Create a new role named **groveInfoRole** as shown.



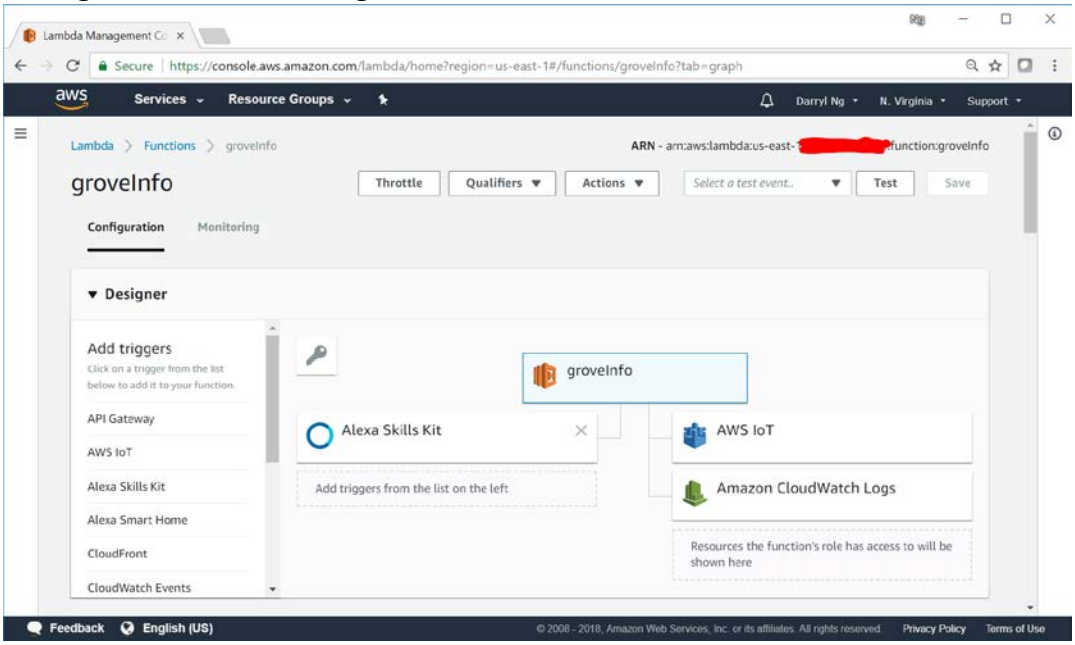
3. Create a new Lambda function by clicking on the '**Create function**' button.



4. Select '**Author from scratch**'. Provide a name for the function and select an existing role you have created earlier.



5. Configure the function designer as shown.



6. Paste the following code on the **Function Code** section of the Lambda function.

```
//Environment Configuration
var config = {};
config.IOT_BROKER_ENDPOINT     = "<Your AWS IoT Endpoint>";
config.IOT_BROKER_REGION       = "<Your AWS IoT Region>";
config.IOT_THING_NAME          = "<Your AWS IoT Thing Shadow>";
config.params                  = { thingName: '<Your PI Hostname>' };

//Loading AWS SDK libraries
var AWS = require('aws-sdk');
AWS.config.region = config.IOT_BROKER_REGION;

//Initializing client for IoT
var iotData = new AWS.IotData({endpoint: config.IOT_BROKER_ENDPOINT});

// Route the incoming request based on type (LaunchRequest, IntentRequest, etc.)
// The JSON body of the request is provided in the event parameter.
exports.handler = function (event, context) {
    try {
        console.log("event.session.application.applicationId=" +
event.session.application.applicationId);

        /*
        if (event.session.application.applicationId !== "amzn1.echo-sdk-
ams.app.c05192e2-4525-429c-8235-3095b4d8fec8") {
            context.fail("Invalid Application ID");
        }
```

```
    */

    if (event.session.new) {
        onSessionStarted({requestId: event.request.requestId}, event.session);
    }

    if (event.request.type === "LaunchRequest") {
        onLaunch(event.request,
            event.session,
            function callback(sessionAttributes, speechletResponse) {
                context.succeed(buildResponse(sessionAttributes, speechletResponse));
            });
    } else if (event.request.type === "IntentRequest") {
        onIntent(event.request,
            event.session,
            function callback(sessionAttributes, speechletResponse) {
                context.succeed(buildResponse(sessionAttributes, speechletResponse));
            });
    } else if (event.request.type === "SessionEndedRequest") {
        onSessionEnded(event.request, event.session);
        context.succeed();
    }
  } catch (e) {
    context.fail("Exception: " + e);
  }
};

/**
 * Called when the session starts.
 */
function onSessionStarted(sessionStartedRequest, session) {
  console.log("onSessionStarted requestId=" + sessionStartedRequest.requestId +",
sessionId=" + session.sessionId);
}

/**
 * Called when the user launches the skill without specifying what they want.
 */
function onLaunch(launchRequest, session, callback) {
  console.log("onLaunch requestId=" + launchRequest.requestId + ", sessionId=" +
session.sessionId);

  // Dispatch to your skill's launch.
  getWelcomeResponse(callback);
}
```

```
/**
 * Called when the user specifies an intent for this skill.
 */
function onIntent(intentRequest, session, callback) {
    console.log("onIntent requestId=" + intentRequest.requestId + ", sessionId=" +
session.sessionId);

    var intent = intentRequest.intent,
        intentName = intentRequest.intent.name;

    // Dispatch to your skill's intent handlers
    if ("GetTemperature" === intentName) {
        getTemperature(intent, session, callback);
    } else if ("GetHumidity" === intentName) {
        getHumidity(intent, session, callback);
    } else if ("GetLight" == intentName) {
        getLight(intent, session, callback);
    } else if ("AMAZON.HelpIntent" === intentName) {
        getHelp(callback);
    } else if ("AMAZON.StopIntent" === intentName || "AMAZON.CancelIntent"
=== intentName) {
        handleSessionEndRequest(callback);
    } else {
        throw "Invalid intent";
    }
}


/**
 * Called when the user ends the session.
 * Is not called when the skill returns shouldEndSession=true.
 */
function onSessionEnded(sessionEndedRequest, session) {
    console.log("onSessionEnded requestId=" + sessionEndedRequest.requestId +
        ", sessionId=" + session.sessionId);
    // Add cleanup logic here
}

// --------------- Functions that control the skill's behavior ----------------------

function getWelcomeResponse(callback) {
    var sessionAttributes = {};
    var cardTitle = "Welcome";
    var speechOutput = "Welcome to Darryl's workshop, Where I communicate with
some sensors in the house running from a raspberry pie," +
    "Would you like to know the temperature or humidity?";
```

```
    var repromptText = "Would you like to know the temperature or humidity?";
    var shouldEndSession = false;

    callback(sessionAttributes, buildSpeechletResponse(cardTitle, speechOutput,
repromptText, shouldEndSession));
}

function getHelp(callback) {
    var sessionAttributes = {};
    var cardTitle = "Help";
    var speechOutput = "Welcome to Darryl's workshop, Where I communicate with
some sensors in the house running from a raspberry pie," +
    "You can ask me what the temperature or humidity is, or ask about the weather for
both.";
    var repromptText = "Would you like to know the temperature or humidity?";
    var shouldEndSession = false;

    callback(sessionAttributes, buildSpeechletResponse(cardTitle, speechOutput,
repromptText, shouldEndSession));
}









function handleSessionEndRequest(callback) {
    var cardTitle = "Session Ended";
    var speechOutput = "Thank you for visiting Darryl's workshop, Have a nice day!";
    var shouldEndSession = true;
    callback({}, buildSpeechletResponse(cardTitle, speechOutput, null,
shouldEndSession));
}

function getTemperature(intent, session, callback) {
  var cardTitle = "Temperature";
  var repromptText = "";
  var sessionAttributes = {};
  var shouldEndSession = true;

  var speechOutput = "";
  var payload = "";

  var temp = 0;
```

```
  iotData.getThingShadow(config.params, function(err, data) {
    if (err)  {
        console.log(err, err.stack); // an error occurred
    } else {
        //console.log(data.payload);          // successful response
        payload = JSON.parse(data.payload);
        temp = payload.state.reported.temperature;
    }

    speechOutput = "The temperature is " + temp + " degrees celcius";
    callback(sessionAttributes, buildSpeechletResponse(cardTitle, speechOutput,
repromptText, shouldEndSession));
  });
}


function getHumidity(intent, session, callback) {
  var cardTitle = "Temperature";
  var repromptText = "";
  var sessionAttributes = {};
  var shouldEndSession = true;

  var speechOutput = "";
  var payload = "";

  var humid = 0;

  iotData.getThingShadow(config.params, function(err, data) {
    if (err)  {
        console.log(err, err.stack); // an error occurred
    } else {
        //console.log(data.payload);          // successful response
        payload = JSON.parse(data.payload);
        humid = payload.state.reported.humidity;
    }

    speechOutput = "The humidity is " + humid + " percent.";
    callback(sessionAttributes, buildSpeechletResponse(cardTitle, speechOutput,
repromptText, shouldEndSession));
  });
}
```

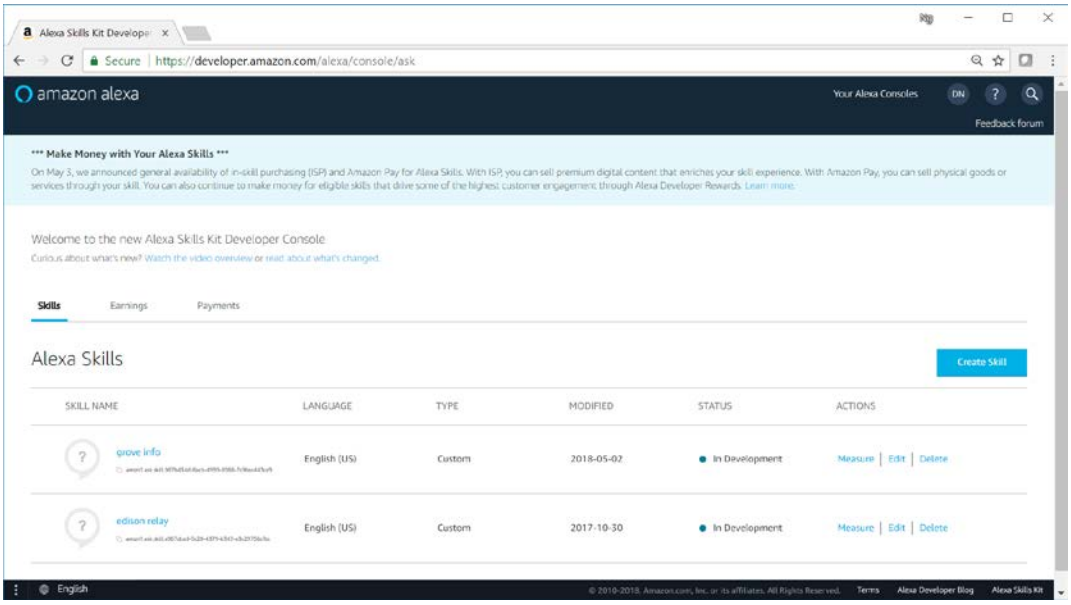
```
function getLight(intent, session, callback) {
  var cardTitle = "Light";
  var repromptText = "";
  var sessionAttributes = {};
  var shouldEndSession = true;

  var speechOutput = "";
  var payload = "";

  var light= 0;

  iotData.getThingShadow(config.params, function(err, data) {
    if (err)  {
        console.log(err, err.stack); // an error occurred
    } else {
        //console.log(data.payload);          // successful response
        payload = JSON.parse(data.payload);
        light = payload.state.reported.light;
    }

    speechOutput = "The light density is " + light + " bright.";
    callback(sessionAttributes, buildSpeechletResponse(cardTitle, speechOutput,
repromptText, shouldEndSession));
  });
}

// --------------- Helpers that build all of the responses ----------------------
function buildSpeechletResponse(title, output, repromptText, shouldEndSession) {
    return {
        outputSpeech: {
            type: "PlainText",
            text: output
        },
        card: {
            type: "Simple",
            title: title,
            content: output
        },
        reprompt: {
            outputSpeech: {
                type: "PlainText",
                text: repromptText
            }
        },
        shouldEndSession: shouldEndSession
```

```
    };
}

function buildResponse(sessionAttributes, speechletResponse) {
    return {
        version: "1.0",
        sessionAttributes: sessionAttributes,
        response: speechletResponse
    };
}
```

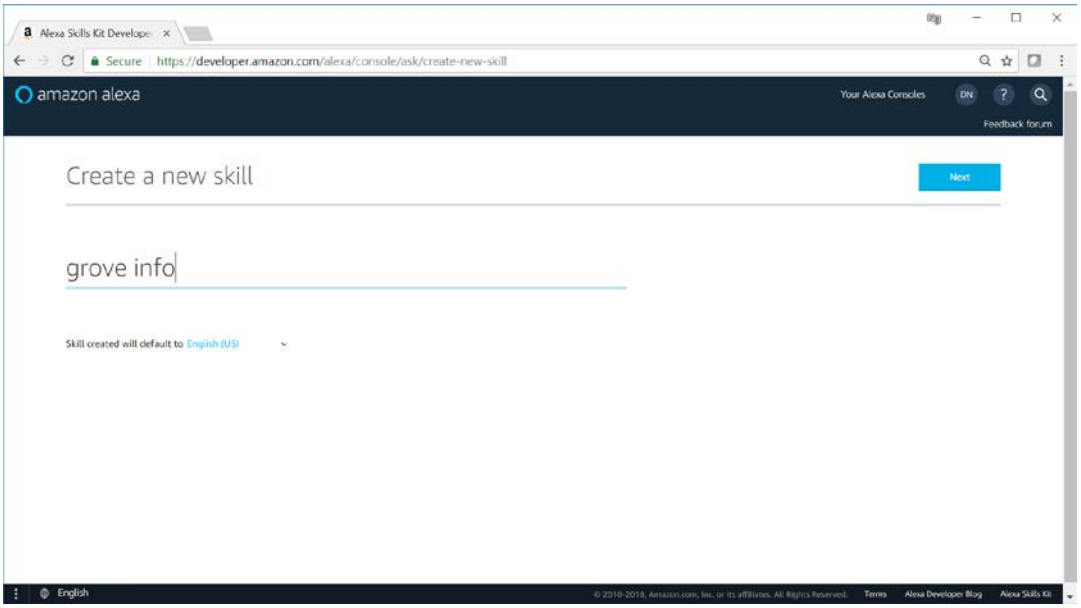
Workshop 7: AWS Alexa Service

1. Navigate to Alexa Console using the following URL https://developer.amazon.com/alexa. Login using your AWS Account ID and password.

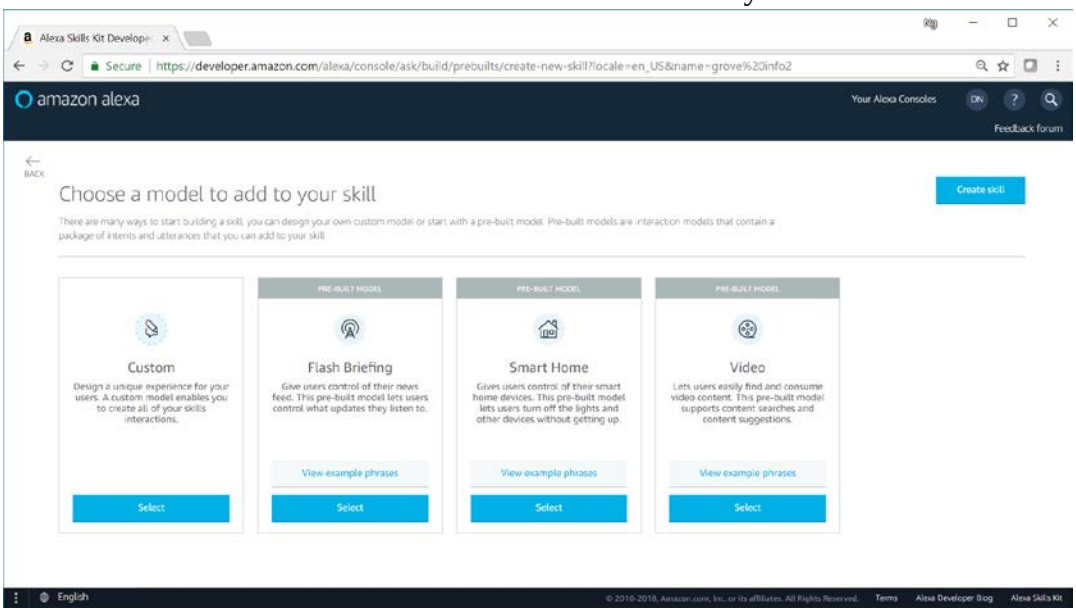
2. Navigate to the Skills console in Alexa Console. Click '**Create Skill**' button to create a new Alexa skill.



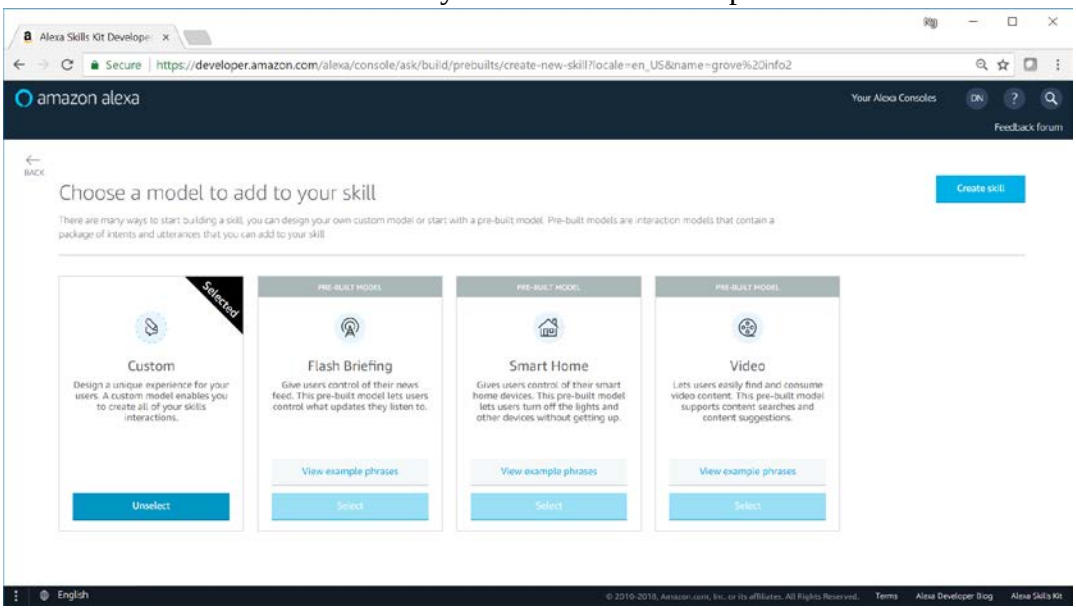
3. Name the skill '**Grove Info**'.



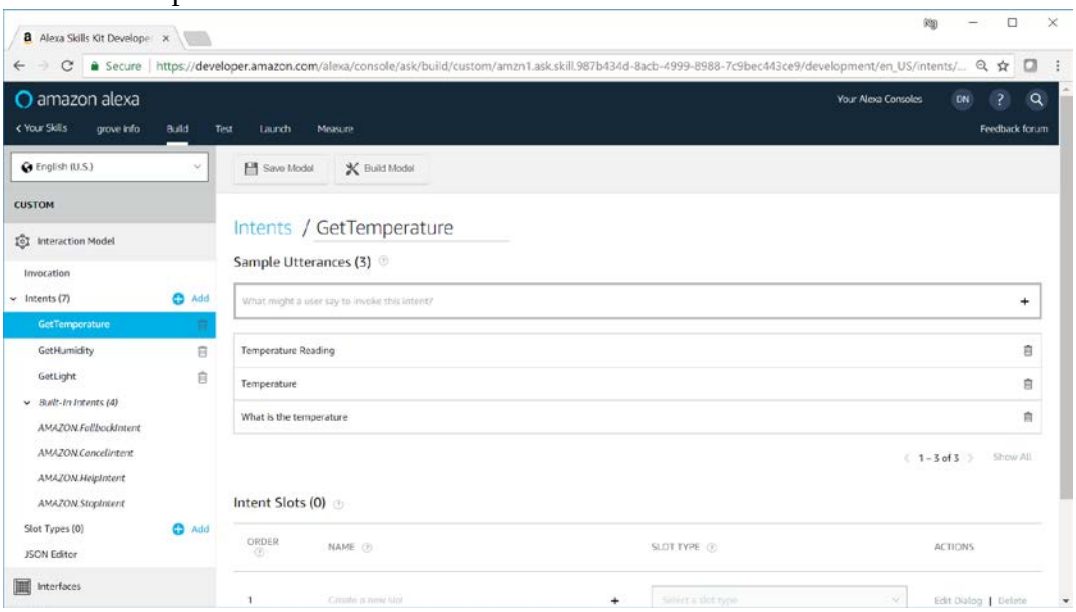
4. Select '**Custom**' when asked to choose a model to add to your skill.



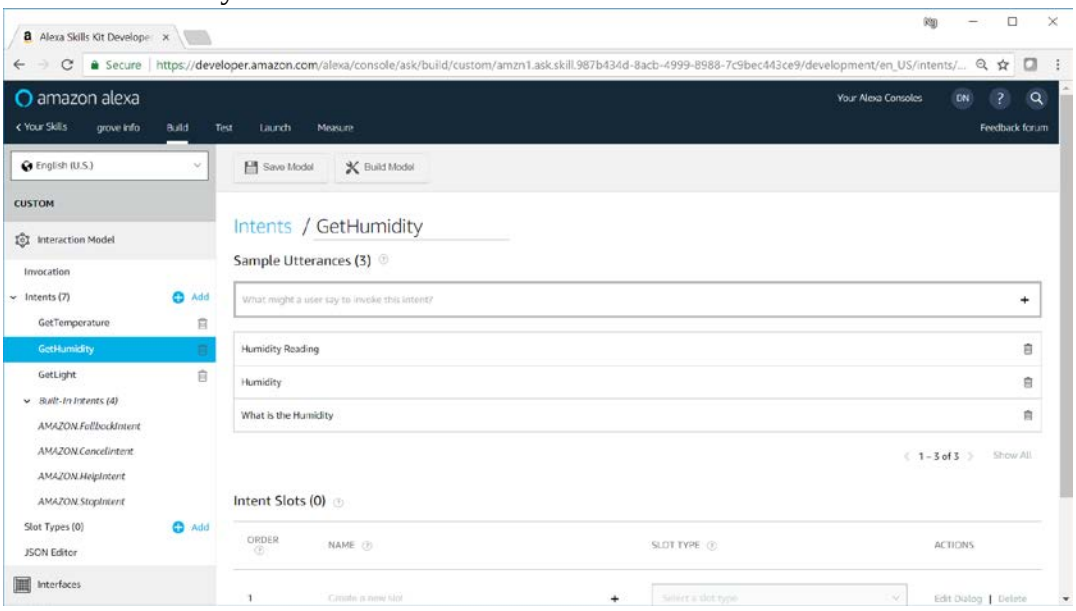
5. Click '**Create Skill**' button when you have selected an option.



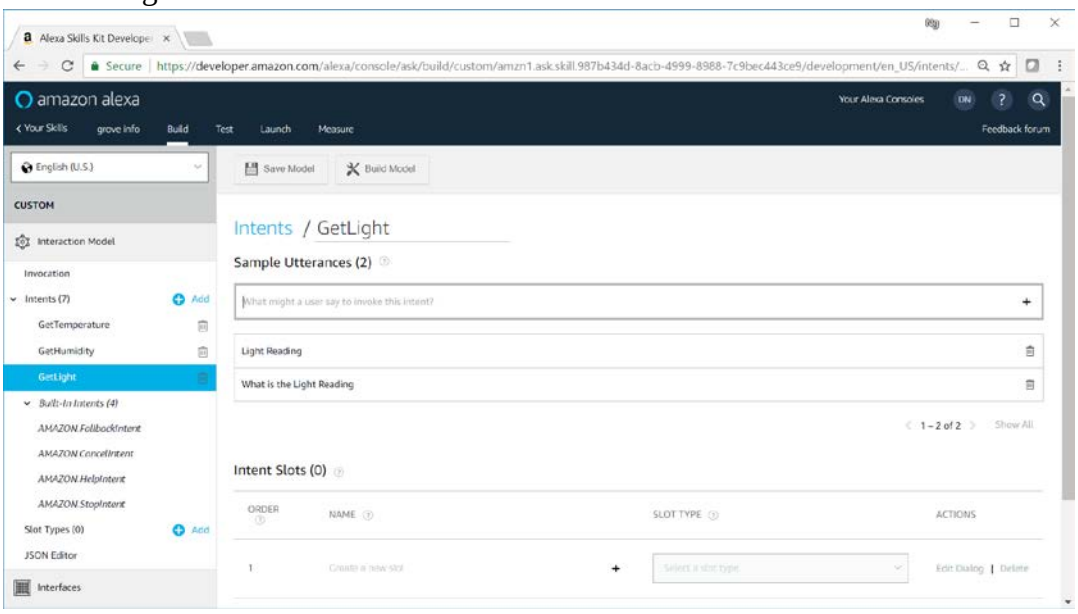
6. Add the Temperature intent as shown.



7. Add the Humidity intent as shown.

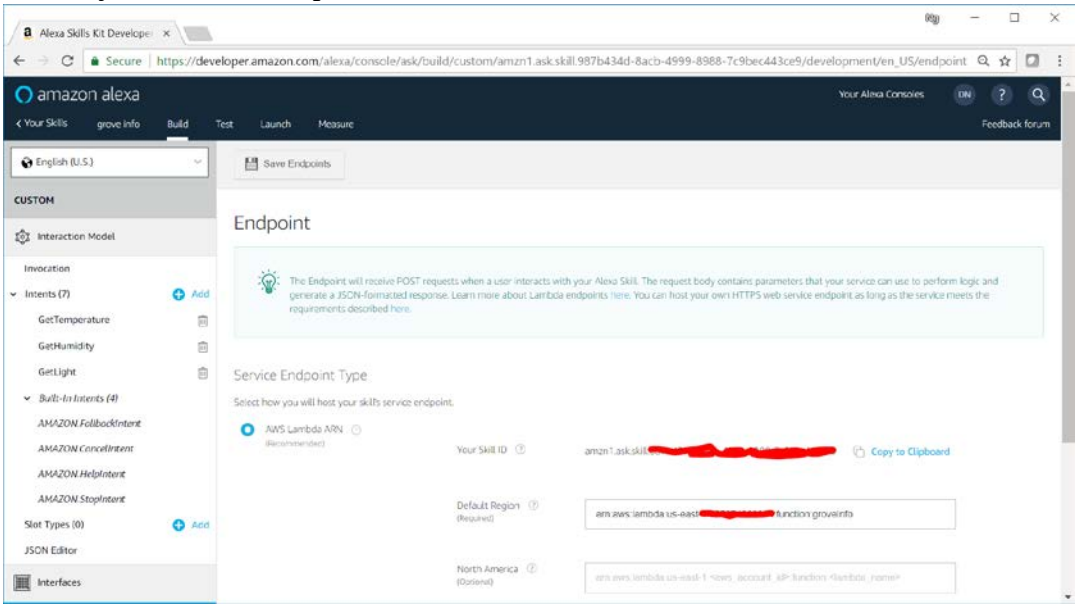


8. Add the Light intent as shown.



9. Save and Build the Model.

10. Define your lambda endpoint as shown.



Your solution is almost ready to run and work under the following architecture.

Workshop 8: Testing your Raspberry PI communication to AWS IoT

In this workshop, you will add python code to support connectivity to AWS IoT service in a python file named **ssdiot01.py** in the project folder **groveproject**.

1. Use the import statement to include the modules required to support connecting your Raspberry Pi to AWS IoT service.

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
import logging
import time
import json
from datetime import date, datetime
```

2. Include in the following AWS configuration information to allow your code to connect to AWS service.

```
#AWS configuration
host = "<Your AWS IoT endpoint>"
rootCAPath = "<Your root CA path and root CA filename>"
certificatePath = "<Your AWS IoT cert path and cert filename>"
privateKeyPath = "<Your AWS private key path and filename>"
clientId = "<Your Raspberry PI name>"
topic = "<Your Pub Sub Topic>"
myMQTTClient = AWSIoTMQTTClient(clientId)
```

3. Write a custom callback function named **customCallback(client, userdata, message)**.

```
# Custom MQTT message callback
def customCallback(client, userdata, message):
    print("Received a new message: ")
    print(message.payload)
    print("from topic: ")
    print(message.topic)
    print("--------------\n\n")
```

4. Write a function named **establishAWS()** that will establish connectivity to AWS.

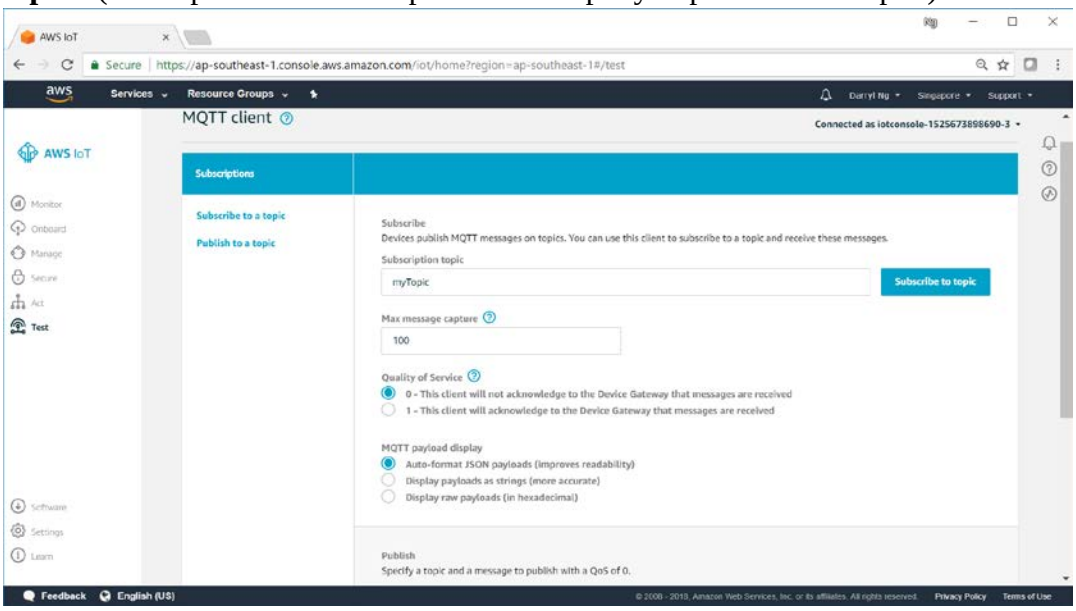
```
def establishAWS():
        myMQTTClient.configureEndpoint(host, 8883)
        myMQTTClient.configureCredentials(rootCAPath,
privateKeyPath, certificatePath)
        myMQTTClient.configureOfflinePublishQueueing(-1)
        myMQTTClient.configureDrainingFrequency(2)
        myMQTTClient.configureConnectDisconnectTimeout(10)
        myMQTTClient.configureMQTTOperationTimeout(5)
```

5. Complete the code as shown.

```
initialFunc()
establishAWS()

while True:
        try:
                [ lightval, tempval, humval ] = readValues()

                resistance = processValue(lightval)
                triggerLight(resistance)
                triggerLCD(tempval, humval)

                now = datetime.utcnow()
                now_str = now.strftime('%Y-%m-%dT%H:%M:%SZ')

                payload = '{ "timestamp" : "' + now_str + '",
"temperature" : ' + str(tempval) + ', "humidity" : ' +
str(humval) + ', "light" : ' + str(lightval) + '}'

                myMQTTClient.connect()
                myMQTTClient.publish("myTopic", payload, 0)
                myMQTTClient.disconnect()

        except (IOError, TypeError) as e:
                print(str(e))
                # and since we got a type error
                # then reset the LCD's text
                setText("")

        # wait some time before re-updating the LCD
        sleep(2)
```

6. Save and execute your code.
   python ssdiot01.py

7. On the AWS IoT console, select **Test** from the menu. Type a topic and '**Subscribe to topic**'. (The topic should correspond to the topic you provided in step 2.)



Your instructor will demonstrate to guide you in sending message from your Raspberry Pi to AWS IoT service via MQTT.

Workshop 9: Modify code for Raspberry PI to AWS IoT for Alexa

In this workshop, you will modify python code to support connectivity to AWS IoT service to communicate with Alexa service in a python file named **ssdiot01.py** in the project folder **groveproject**.

1. Use the import statement to include the modules required to support connecting your Raspberry Pi to AWS IoT service.

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
import logging
import time
import json
from datetime import date, datetime
```

2. Modify AWS configuration information to allow your code to connect to AWS service.

```
#AWS configuration
host = "<Your AWS IoT endpoint>"
rootCAPath = "<Your root CA path and root CA filename>"
certificatePath = "<Your AWS IoT cert path and cert filename>"
privateKeyPath = "<Your AWS private key path and filename>"
clientId = "<Your Raspberry PI name>"
topic = "<Your Pub Sub Topic>"
myMQTTClient = AWSIoTMQTTShadowClient(clientId)
```

3. Modify your function named **establishAWS()** as follows.

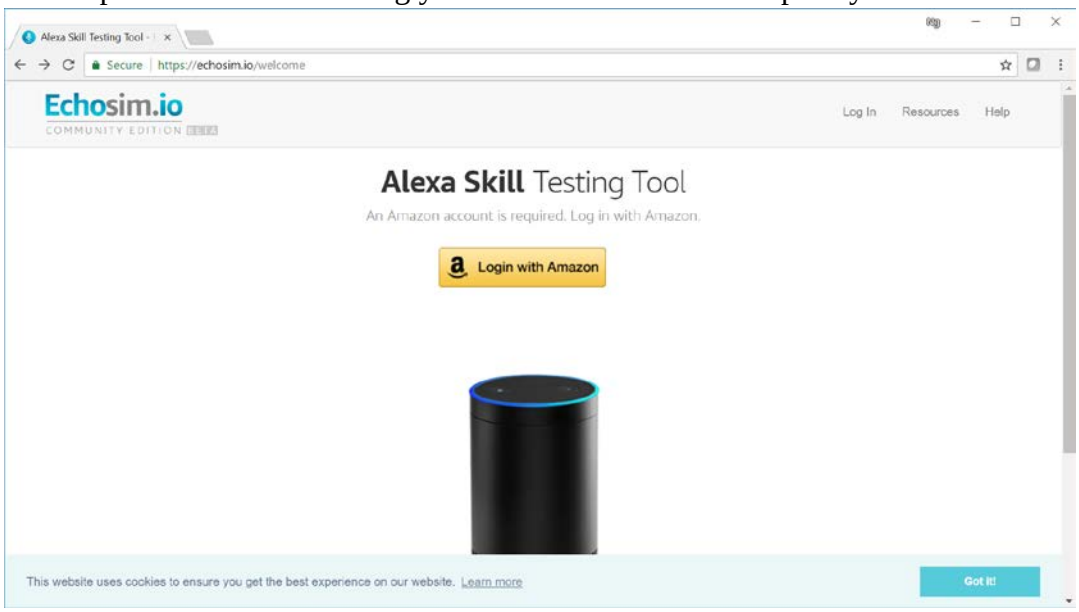
```
def establishAWS():
        myMQTTClient.configureEndpoint(host, 8883)
        myMQTTClient.configureCredentials(rootCAPath,
privateKeyPath, certificatePath)
        # myMQTTClient.configureOfflinePublishQueueing(-1)
        # myMQTTClient.configureDrainingFrequency(2)
        myMQTTClient.configureConnectDisconnectTimeout(10)
        myMQTTClient.configureMQTTOperationTimeout(5)
```

4. Complete the code as shown.

```
initialFunc()
establishAWS()

while True:
        try:
                [ lightval, tempval, humval ] = readValues()

                resistance = processValue(lightval)
                triggerLight(resistance)
                triggerLCD(tempval, humval)

                now = datetime.utcnow()
                now_str = now.strftime('%Y-%m-%dT%H:%M:%SZ')

                payload = "{ \"state\" : { \"reported\" :
{ \"timestamp\" : \"%s\", \"temperature\" : \"%s\",
\"humidity\" : \"%s\", \"light\" : \"%s\" } } }" % (now_str,
str(tempval), str(humval), str(lightval))

                myMQTTClient.connect()
                myDeviceShadow =
myMQTTClient.createShadowHandlerWithName("darrylpi", True)
                myDeviceShadow.shadowUpdate(payload, None, 5)
                myMQTTClient.disconnect()

        except (IOError, TypeError) as e:
                print(str(e))
                # and since we got a type error
                # then reset the LCD's text
                setText("")

        # wait some time before re-updating the LCD
        sleep(2)
```

5. Save and execute your code.
   python ssdiot01.py

6. You may navigate to https://echosim.io/welcome and login with your AWS Account ID and password to start testing your Alexa service with Raspberry Pi.



~ End of Workshop ~