**Creating an Alexa voice controlled IoT using a Raspberry Pi**

**Thanks** to Jeff Nunn at Amazon upon whose work a lot of this is based.

You can follow his videos here:
https://www.youtube.com/watch?v=40zklUynHGc&list=PL2KJmkHeYQTNKbeNmYxs-CY3AhPJcl61U&index=1

The videos for this start at: https://youtu.be/GU1Dkha5z4o

Please work through them all in order.

**Objective**

The purpose of this is to be able to control some hardware using voice control with an Alexa using publish and subscribe (pub/sub). Here I use a Raspberry Pi with a Sense Hat as a display.

There are three ways to control an IoT from Alexa, 1) paring via Bluetooth, 2) using pub/sub, and 3) a stand-alone voice controlled 'thing'

We will be controlling our robot over the internet using 2) pub/sub. If you want the others, I suggest you look at https://github.com/alexa/Alexa-Gadgets-Raspberry-Pi-Samples

Stages

1. Create an Internet Of Things (IoT) for our device. This will give us certificates and policies to communicate using MQTT messaging protocol
2. We will edit and run the python code on the Pi
3. We will send MQTT messages to the Pi using IoT pub/sub to check that it obeys the commands.
4. Create a lambda function and edit and upload the lambda code
5. Set up our Alexa Skill voice interface
6. Test it by talking to our Alexa

**Hardware:**

You will of course need an Amazon Alexa

I used a Raspberry Pi 4 with a Sense hat. It looks like this:

If you don't have a SenseHat, you can modify the code to light an LED and (later) detect a switch press.

The Raspberry Pi needs setting up. You will need an operating system if you don't have one, and it needs to be connected to your wi-fi and ssh needs setting up. I also use putty, VNC and WinSCP to transfer files between the PC and RPi.

If you don't know how to do all this, take a look at the following video.

https://www.youtube.com/watch?v=uLwj4Wj7pRI

You don't need to do the section where you connect via your smartphone if you can find out the RPi's wifi address (e.g. 192.168.1.53) from your router.

You will need to enable VNC in the RPi settings.

The quick set-up instructions are and the end of this.

**Software.**

An Amazon Developer Account. If you don't already have a developer account, create one.

You will need accounts with Amazon developer account (developer.amazon.com/) and AWS (aws.amazon.com) – this asks for a credit card, but for these purposes you should not be charged and. It will also help if you have previously created an Amazon SDK skill too.

I've used US (East Virginia) for my server.

We will communicate using Publisher /Subscribe (AKA Pub/Sub)

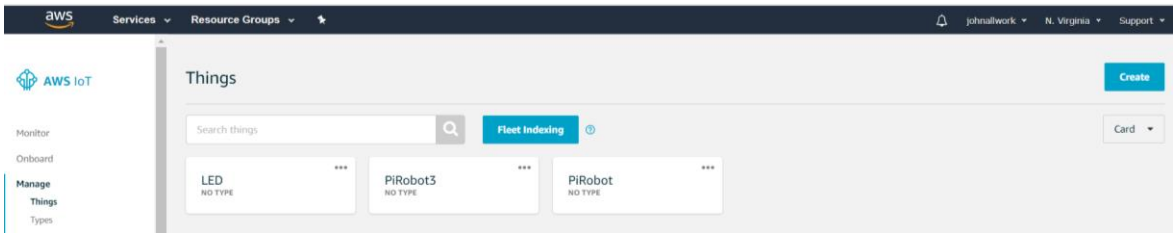
Our Pi will subscribe to and listen for messages sent to it via our Alexa voice command, which will run some python lambda code that sends (publishes) the messages (e.g. turn red, display hello, sparkle etc.)

We will use MQTT to do this. It's a lightweight messaging protocol for devices.

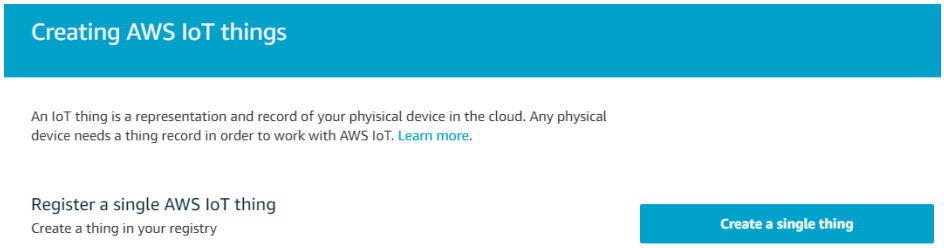
Let's start.

**1. Create our 'Thing'**

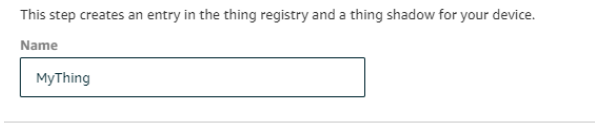
Go to the AWS IoT console (https://console.aws.amazon.com/iot/) and select Manage > Things. Click Create



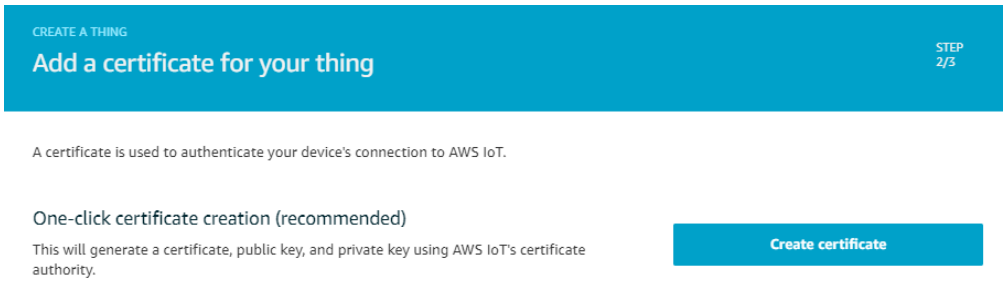

And select 'Create a Single Thing'




Give it a name (MyThing), ignore the other boxes and click Next



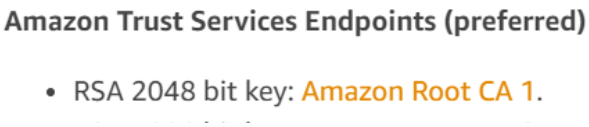

Now we need a certificate. This will authenticate our thing with AWS IoT. Click **Create certificate**




Download the certificates

And the root CA for IoT. Click the link, then click CA certificates

And select RSA 2048 bit key: Amazon Root CA1

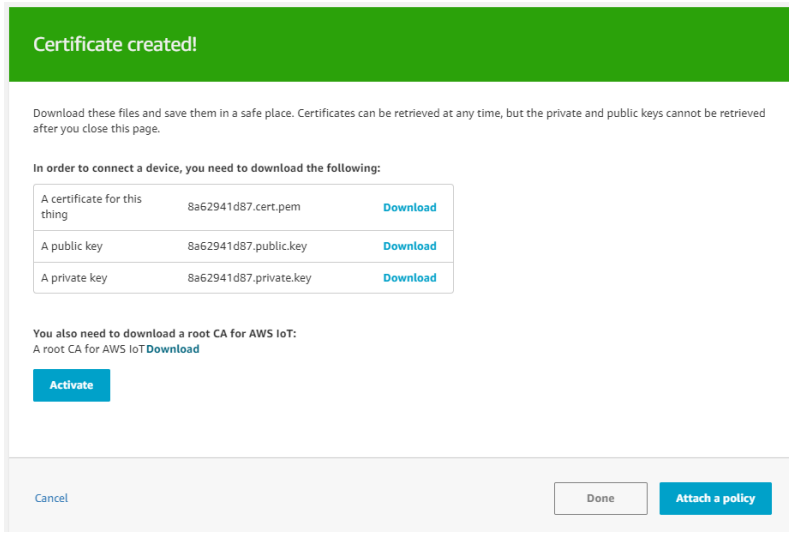



This starts with

```
-----BEGIN CERTIFICATE-----
```

Copy all of this into Notepad and save as AmazonRootCA1.crt

We will transfer (ssh) these files to our RPi in a while. I haven't tried it, but if you log on to AWS using the browser on your Pi, you won't have to transfer them from your PC

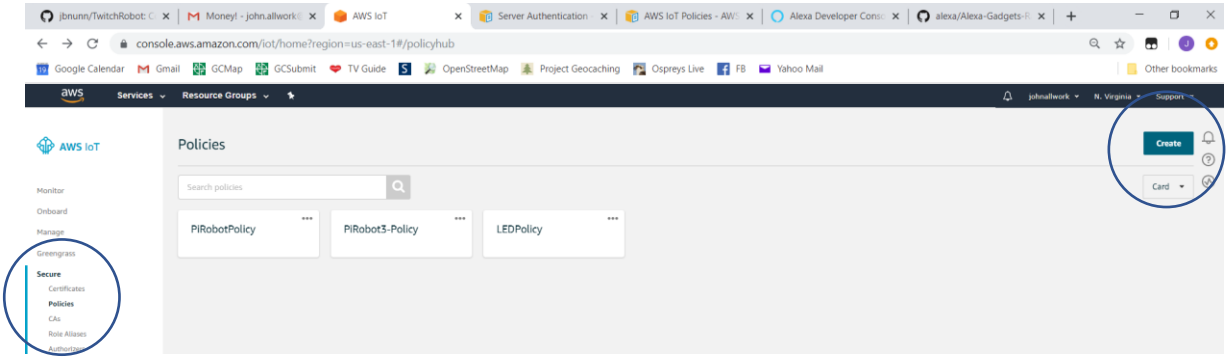


Click **Activate** and **Done**

Now we need to attach a policy to the certificates. A policy control access to the AWS IoT.

First, create your policy. In the IoT menu, on the left-hand side, click **Secure > Policies**

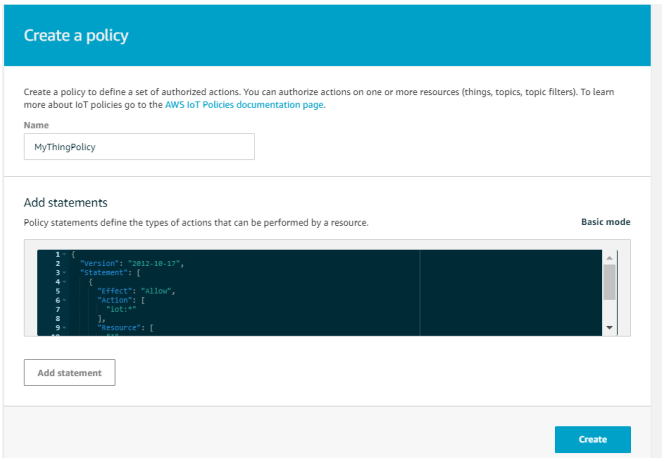
Click **Create** and give it a name (e.g. MyThingPolicy)




Click Advanced and edit the provided policy to the following (no, I don't know what it all means either)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

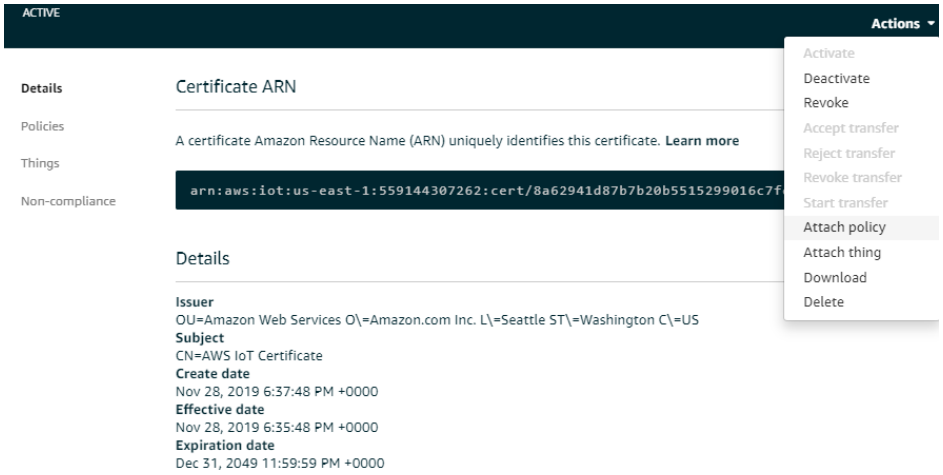
And click **Create** to create your policy



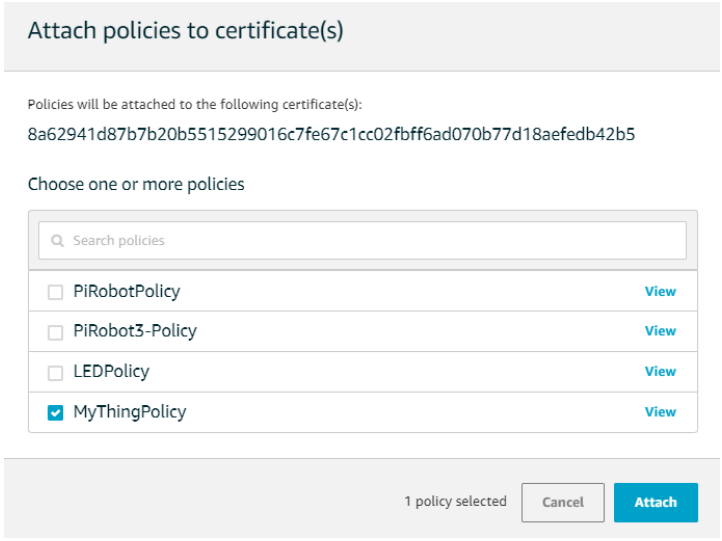
Now we need to attach the policy to the certificates.

Return to AWS IoT > Secure > Certificates

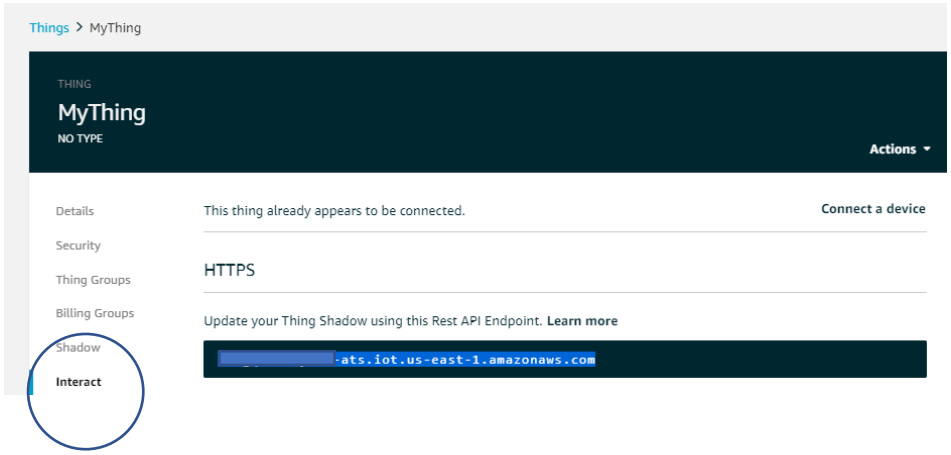
Choose your policy > Action Attach policy




If you have more than one policy created, select the one you want and click **Attach**




We also need an endpoint address. To find this, go to AWS IoT > Manage > Things > select MyThing, then **Interact** and copy the https information.

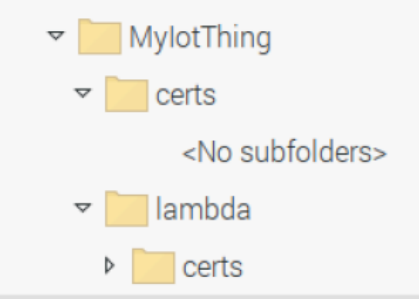


It's something like: `abcdefghijk123-ats.iot.us-east-1.amazonaws.com`

This will be the **endpoint** in the listener code.

### Transfer the files

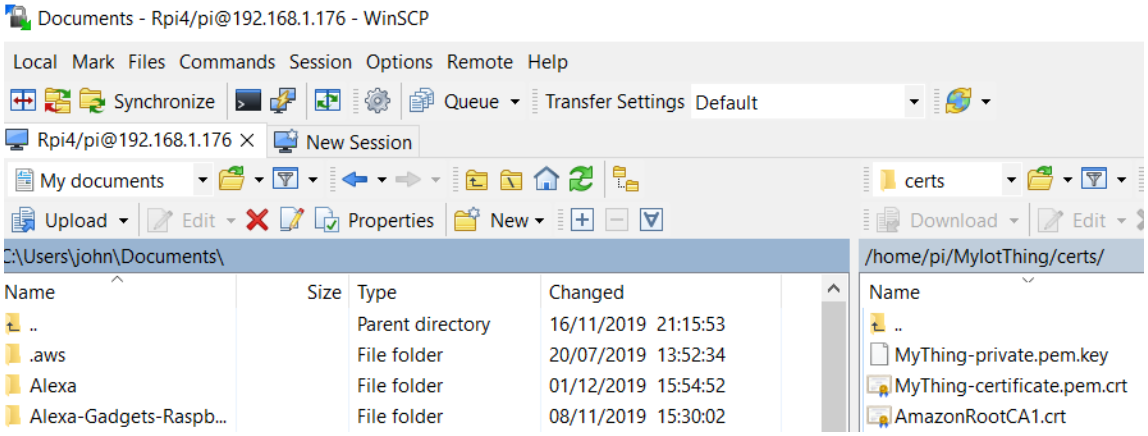
Create a folder on your Pi called MyThing. Within that create folders called **certs** and **lambda**. Later we will also need a folder called **certs** in the lambda folder, so you can also create that now.




Transfer the certificates to your RPi **MyThing/certs** folder. I use WinSCP to do this.

Rename the certs to MyThing-private.pem.key, MyThing-certificate.pem.crt and use AmazonRootCA1.crt. (You don't need the public cert)

If you do this, you won't have to change them in the program.



## 2. Edit and run the python code

### Log into your Pi

If you haven't done so, install the libraries for your Sensehat (or whatever hardware you have) e.g:

```
sudo apt-get install sense-hat
```

(see : https://projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat)

We need the files for the AWSIoT, so install them:

```
sudo pip install AWSIotPythonSDK
```

and you might as well update everything:

```
sudo apt-get update
```

Now we need a listener program: Type this in or copy it and save it in the MyIotThing folder as **listener.py**:

```
#!/usr/bin/python3

import sys
import signal
import time
import json

from sense_hat import SenseHat
sense = SenseHat()

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

createMQTTClient = AWSIoTMQTTClient("MyThing")
createMQTTClient.configureEndpoint( 'abcdefghijk123-ats.iot.us-east-
1.amazonaws.com ', 443)

# Check these certificate names
createMQTTClient.configureCredentials("/home/pi/MyThing/certs/AmazonRootCA1.crt"
, "/home/pi/MyThing/certs/MyThing-private.pem.key",
"/home/pi/MyThing/certs/MyThing-certificate.pem.crt")

createMQTTClient.configureOfflinePublishQueueing(-1)  # Infinite offline Publish queueing
createMQTTClient.configureDrainingFrequency(2)  # Draining: 2 Hz
createMQTTClient.configureConnectDisconnectTimeout(10)  # 10 sec
createMQTTClient.configureMQTTOperationTimeout(5)  # 5 sec

createMQTTClient.connect()
print("Connected")

def unsubscribe_topics():
    """Unbsubscribes from AWS IoT topics before exiting
    """
    print("Unsubscribing")

    topics = [
        '/myPi
```

```
    ]

    for topic in topics:
        createMQTTClient.unsubscribe(topic)

# Interrupt Handler useful to break out of the script
def interrupt_handler(signum, frame):
    unsubscribe_topics()
    sys.exit("Exited and unsubscribed")

# Custom MQTT message callbacks
def driveCallback(client, userdata, message):
    print(f"Received {message.payload} from {message.topic}")
    payload = json.loads(message.payload)
    command = payload['directive']
    print(f"Processing command: {command}")

    if command == "hello":
        sense.show_message("Hi there!")
    elif command == "red":
        sense.clear((255, 0, 0))
    elif command == "stop":
        sense.show_message("Stopped!")
    else:
        print("Command not found")

# Subscribe to topics
createMQTTClient.subscribe("/myPi", 1, driveCallback)
print("Listening on /myPi")

while True:
    signal.signal(signal.SIGINT, interrupt_handler)
    time.sleep(1)

unsubscribe_topics()
```

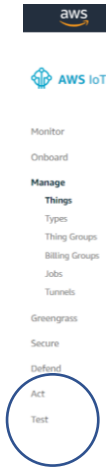
Don’t forget to get the correct path and names for the certificates and also your **endpoint** in the line of code:

**createMQTTClient.configureEndpoint('abcdefgh123345-ats.iot.us-east-1.amazonaws.com', 443)**

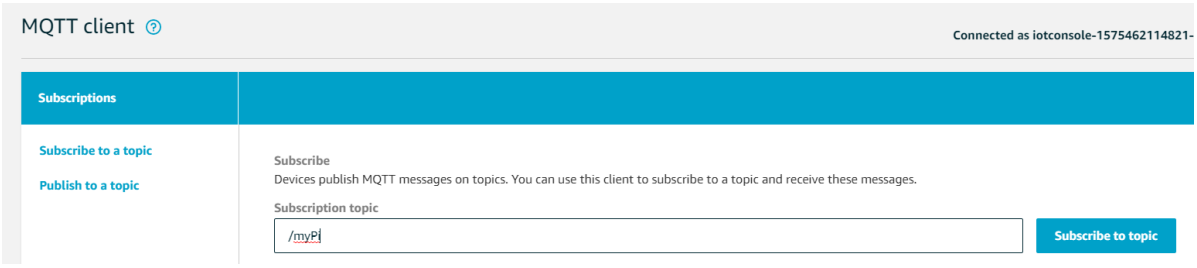
Run the code (type: **python3 listener.py**) and check that it works. If it fails, it will be a problem with your endpoint or certificate.

### 3. Send MQTT messages

On your PC log into AWS console and go to IoT console > Test

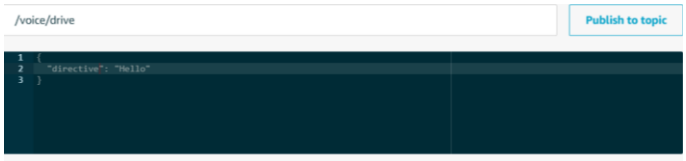



Enter /myPi and click '**Subscribe to topic**'




You can now enter commands. Try entering

```
{
  "directive": "hello"
}
```




And click **Publish to topic**

Watch your RPi. It should display the message Hi World!, (or execute the code you had for the 'hello' command).

If it doesn't work check that you have entered the correct directive, 'hello' not 'Hello'.

### Change the code

You can skip this bit if you want and go to 4). Let's change the code, say we want the device to sparkle for a specific amount of time. We will pass more information, let's call it 'data'. This is sent in a dictionary, which means in future, we pass multiple key/value pairs - a named variable and its value.

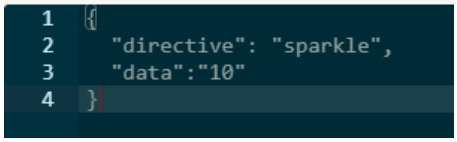
Change the code, so that the driveCallback code becomes:

```
def driveCallback(client, userdata, message):
    print(f"Received {message.payload} from {message.topic}")
    payload = json.loads(message.payload)
    command = payload['directive']
    sparkleTime = payload['data']
    print(f"Processing command: {command}")

    if command == "hello":
        sense.show_message("Hi there!")
    elif command == "red":
        sense.clear((255, 0, 0))
    elif command == "sparkle":
        sense.clear((0,0,0))
        timeStart = time.time()
        timeNow = time.time()
        while timeNow-timeStart < sparkleTime :
          x = randint(0, 7)
          y = randint(0, 7)
          r = randint(0, 255)
          g = randint(0, 255)
          b = randint(0, 255)
          sense.set_pixel(x, y, r, g, b)
          sleep(0.1)
          timeNow= time.time()
    elif command == "stop":
        sense.show_message("Stopped!")
    else:
        print("Command not found")
```

and publish the following to /myPi:

```
{
  "directive": "sparkle",
  "data": "10"
}
```




And the device should sparkle for a short time.

Try just sending the directive on its own, without any data and see what happens.

(Does your listener.py code error?)

Note: The Alexa skill we will create doesn't (yet) pass the time slot to our device. That's further work.

### 4. Create the Lambda function code and upload

Now that we know the Pi code is working and obeying commands sent to the /myPi topic, we need our Alexa Skill to listen to our user, and our lambda code to send the commands to the Pi

Ref: https://docs.aws.amazon.com/lambda/latest/dg/lambda-python-how-to-create-deployment-package.html

We will create our lambda function on the Pi and Zip that with the required dependencies (libraries) and upload it all to AWS site where it will be executed.

Log back into your Pi. If you haven't already done so, create a folder called **lambda** in the MyIotThing folder, and copy the lambda_function.py code into the lambda folder (this is at the end). Create a certs folder within the lambda folder and copy the certs into that folder.

In the lambda_function code, check the certs addresses are correct and **edit the** abcdefgh123345 **endpoint** address to match yours:

createMQTTClient = AWSIoTMQTTClient("MyThing")

createMQTTClient.configureEndpoint('**abcdefgh123345**-ats.iot.us-east-1.amazonaws.com', 443)

createMQTTClient.configureCredentials('./certs/AmazonRootCA1.crt','./certs/**MyThing-private**.pem.key', './certs/ **MyThing-certificate**.pem.crt')

Note that the function_lambda code requires the certs to be in a folder called certs (in the lambda folder)

We now need to bundle up the lambda_function.py code and certificates with all the required libraries (dependencies) and upload that to AWS lambda.

Install the dependencies if you haven't already have done this

```
cd lambda
$ pip install --target ./package AWSIoTPythonSDK
$ pip install --target ./package ask-sdk-core
```

Zip the dependencies

$ cd package

$ zip -r9 ${OLDPWD}/lambda_function.zip .

(the full stop at the end is important)

If you get the error message: "zip command not found", install zip using:

```
sudo apt-get install zip unzip
```

**Add your Lambda code to the zip file**

$ cd $OLDPWD

$ zip -g lambda_function.zip lambda_function.py

If you get a zip warning: name not matched: lambda_function.py, check that the function_lambda.py is in the lambda folder

**Add your certs to the zip file**

If you haven't done so, create a certs folder in the lambda folder and copy the certificates to the /lambda/certs/ folder. Add these to your zip file:

$ zip -g lambda_function.zip certs/*

You could create a batch file for all this of course.

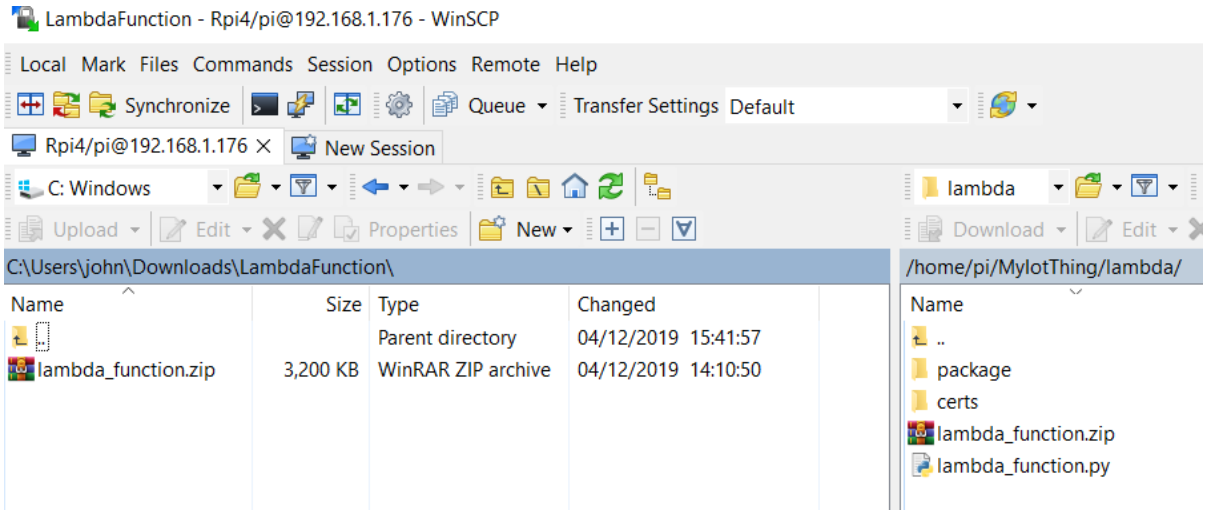
We should now have a lambda_function.zip file that contains our program, the certs and dependencies

**Upload the code to your Lambda function**

If you have aws installed on your Pi, you can do this in one statement.

> You can install aws using `sudo apt-get install awscli`
>
> Note: Replace YOURAWSFUNCTION below with the name of your AWS Lambda function.
>
> $ aws lambda update-function-code --function-name YOURAWSFUNCTION --zip-file fileb://lambda_function.zip

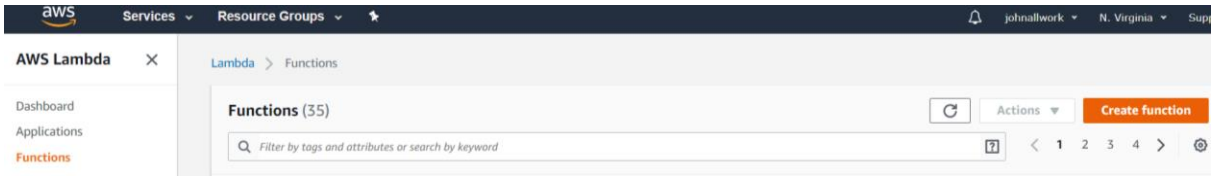
If not use WinSCP to transfer the lambda_function.zip file to your PC:




Now we will create the AWS lambda function. Log onto the AWS lambda console

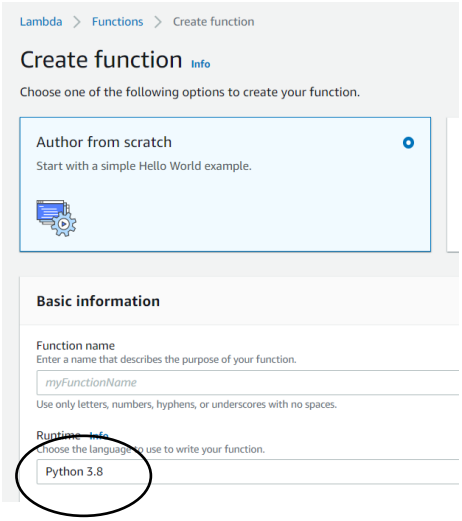
[console.aws.amazon.com/lambda/home](console.aws.amazon.com/lambda/home)

Check that you have the correct region (e.g.us-east-1)

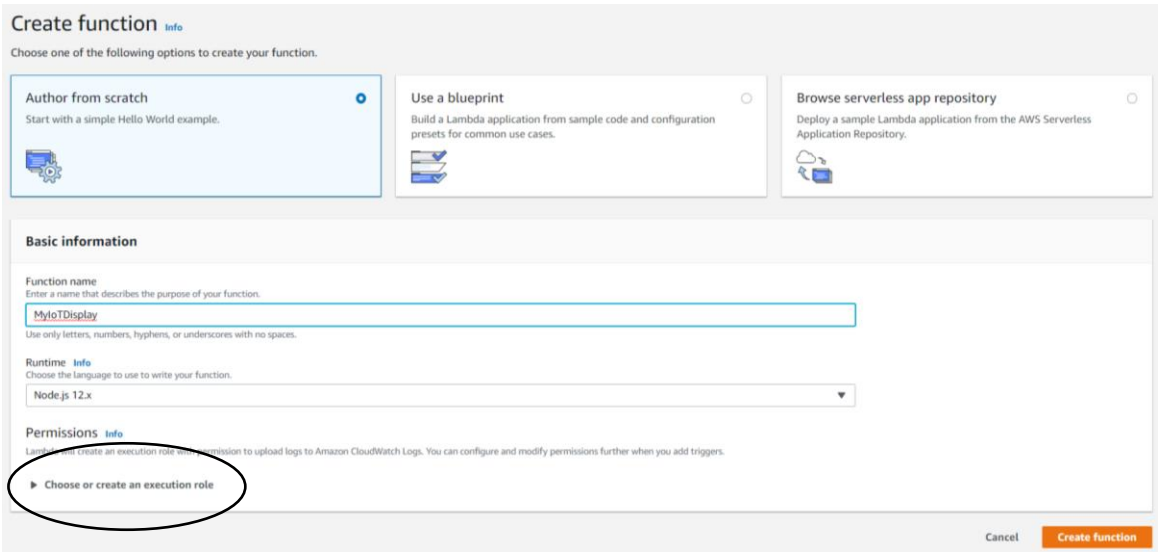



Click **Create function**

Choose "**Author from Scratch**", give your Lambda function a name (e.g. MyThing), and choose the "Python 3.8" or later runtime.

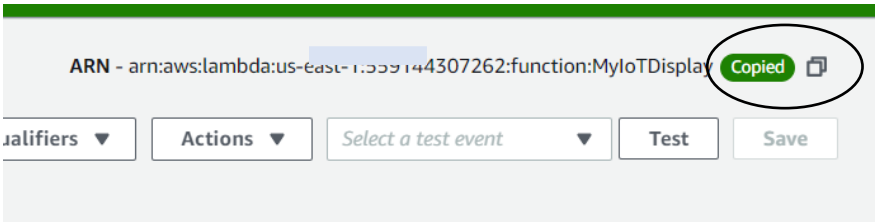



Under "Permissions", choose "Create a new role with basic Lambda permissions."




Click **Create Function**.

Make note of the Amazon Resource Name (ARN) at the top of the page. You'll need this later.



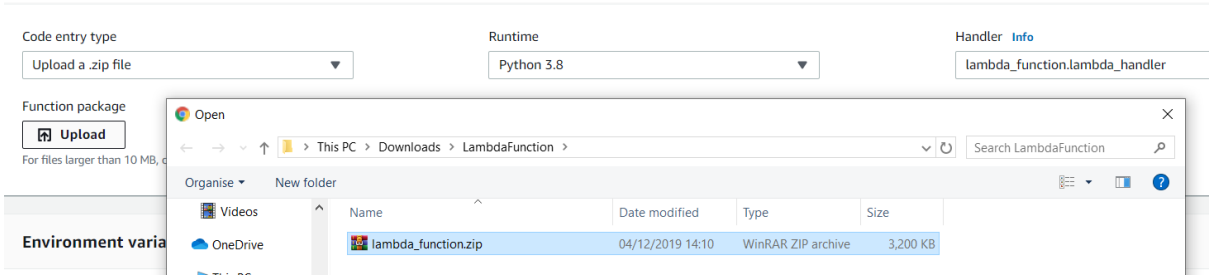

We will add a trigger from our Alexa Skill later.

Now we will upload the lambda function code.

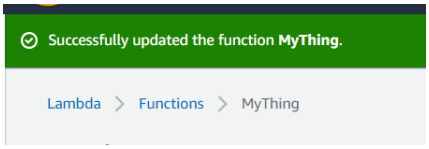
Open your function and choose Upload a zip file:

**Check the Runtime is Python 3.8** (or similar) and the Handler_info is lambda_function.lambda_handler. If not start a new function and select python

click Upload, select your file




It should upload




Click **Save**

N.b. If you used "AWS_IOT_ENDPOINT" in your code, e.g:

createMQTTClient.configureEndpoint(os.environ['AWS_IOT_ENDPOINT'], 443)

you will need to add that in the environment variables: use the key "AWS_IOT_ENDPOINT" and your endpoint value , e.g. 1233456abdef-ats.iot.us-east-1.amazonaws.com

We now need to set up our Alexa skill, tell that skill about this lambda and finally add an Alexa skill trigger and link the lambda to the skil.

**5. Set up the Alexa Skill**

Now we need to do the voice control skill.

Our voice commands will be something like this:

You: “Alexa Open my thing”

Alexa: “OK – you can say hello, turn the display red or sparkle”

You: “Hello” (or Hi, or Hi there, or similar)

Alexa: “Hello” (and Hello world! is displayed)

You: “Turn red” (or red, or glow red, or similar)

Alexa: “OK – red displayed” (Pi turns red)

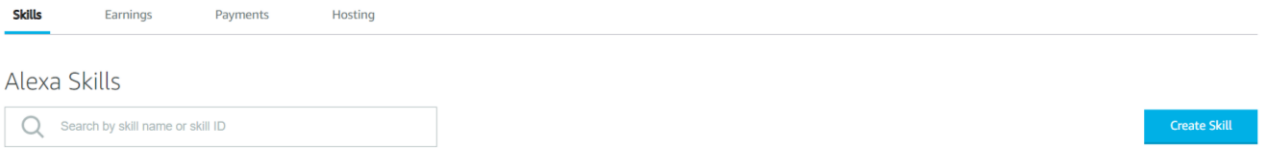
You: “Sparkle” (or twinkle or similar)

Alexa: “Ok, sparkling”

You: “Stop”

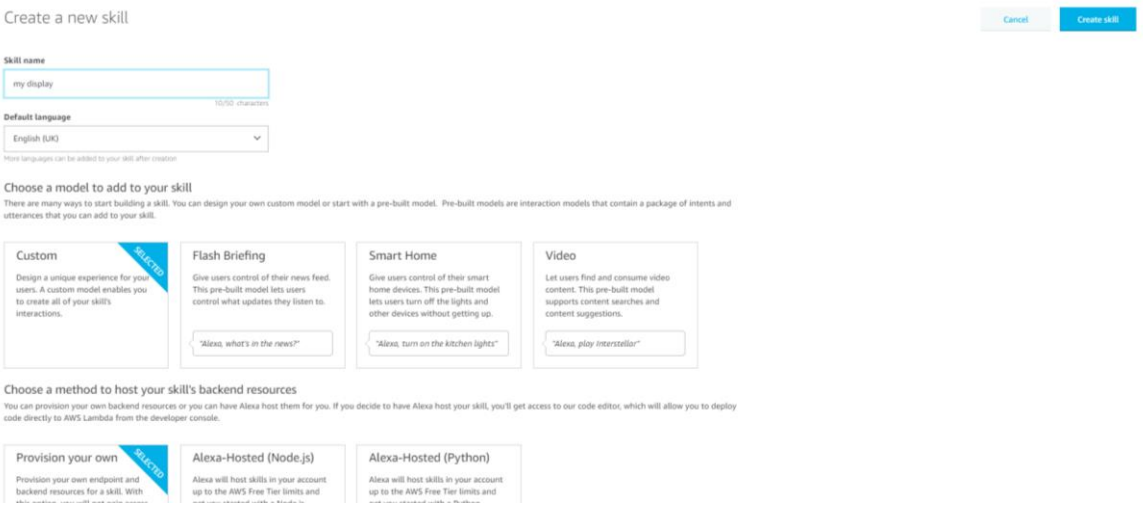
Program terminates

Log into developer.amazon.com > development console > Alexa Skills Kit

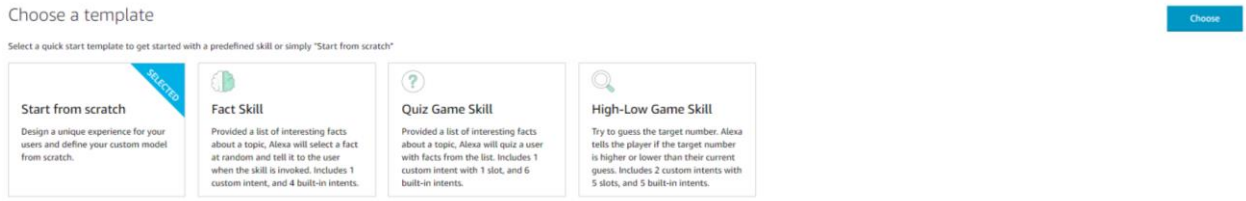


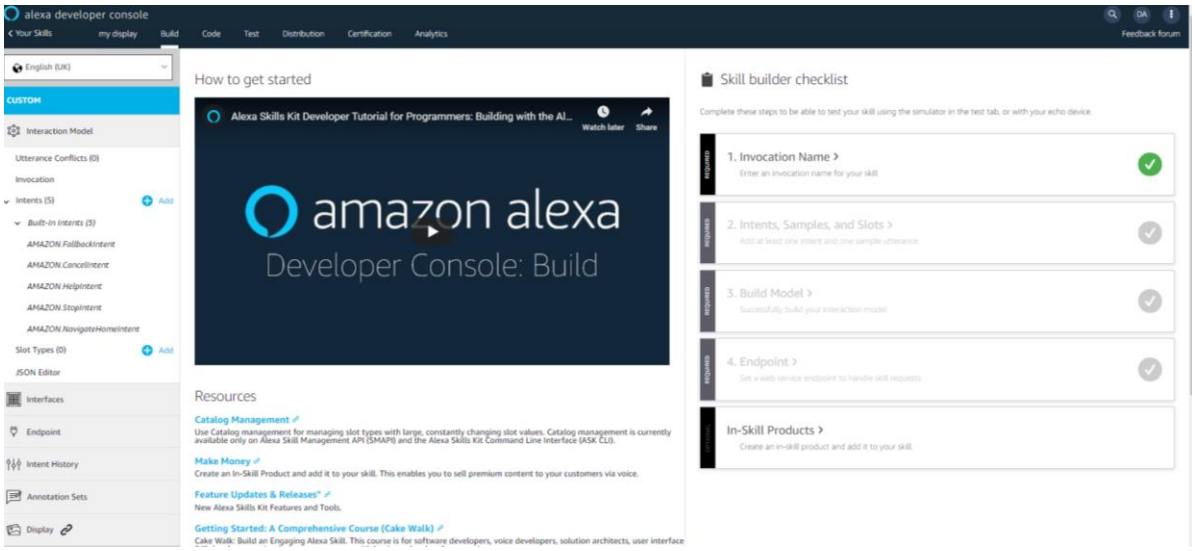

Click **Create Skill**

Enter a Skill name (e.g. “my display thing”), choose default Custom model and Provision your own code and click **Create Skill**




Choose Start from scratch and click **Choose**.



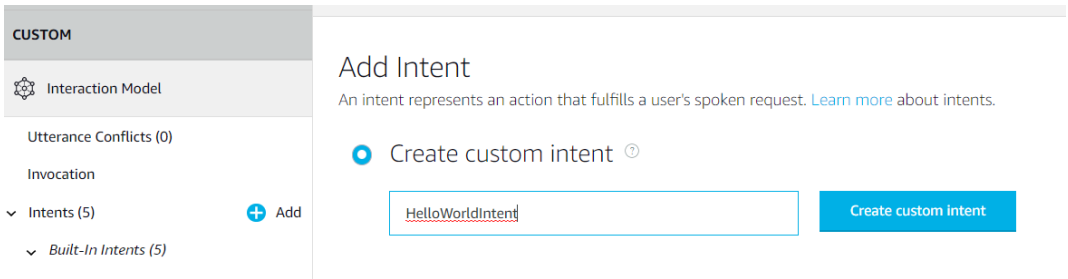




I suggest that you go through the following steps. But you can skip them and copy and paste the JSON code at ** later on.

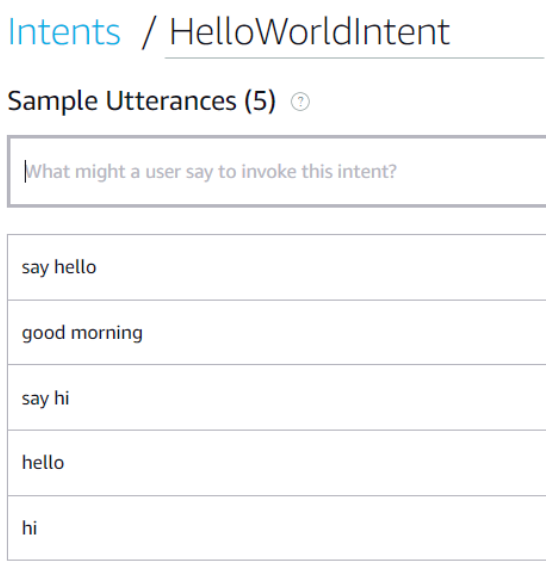
We will now add our intents. This will reflect the commands in our program (hello, red and sparkle) and will be called by our lambda code which will we write after creating our skill.

We will create intents called ‘HelloWorldIntent’, ‘RedIntent’ and SparkleIntent’

Click the + sign to add an intent, name it: HelloWorldIntent.  And click **Create custom intent**



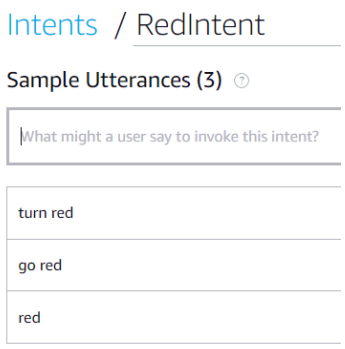

Now enter some utterances – what your user might say to invoke this intent: hello, hi, hi there, good morning etc. Add these one at a time. You should have something like this:




Click Save Model at the top of the page.

Now add the other intents:

RedIntent - utterances: ‘red’, ‘go red’, ‘turn red’ etc




.

SparkleIntent - utterances: ‘sparkle’, ‘twinkle, ‘go mad’ etc. Add these intents

We might want to sparkle for a specific length of time, but we’ll look at that later

Save and build your model.

** You can copy and paste the following into the JSON editor if you prefer:

```
{
    "interactionModel": {
        "languageModel": {
            "invocationName": "my raspberry thing",
            "intents": [
                {
                    "name": "AMAZON.FallbackIntent",
                    "samples": []
                },
                {
```

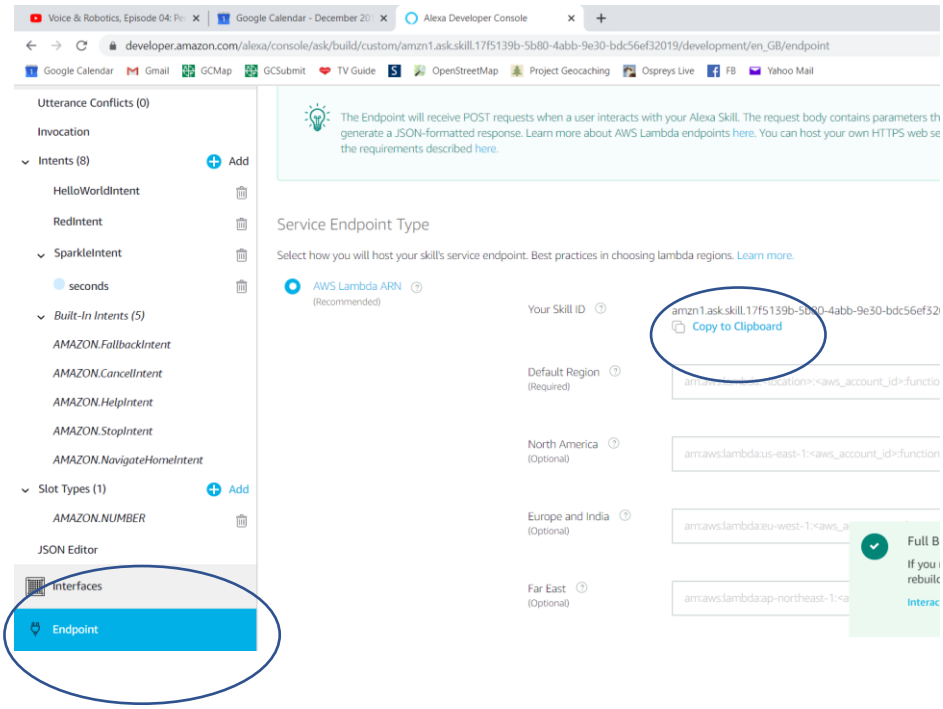
```
            "name": "AMAZON.CancelIntent",
            "samples": []
        },
        {
            "name": "AMAZON.HelpIntent",
            "samples": []
        },
        {
            "name": "AMAZON.StopIntent",
            "samples": []
        },
        {
            "name": "AMAZON.NavigateHomeIntent",
            "samples": []
        },
        {
            "name": "HelloWorldIntent",
            "slots": [],
            "samples": [
                "hi there",
                "good morning",
                "hello",
                "hi"
            ]
        },
        {
            "name": "RedIntent",
            "slots": [],
            "samples": [
                "red",
                "turn red",
                "go red"
            ]
        },
        {
            "name": "SparkleIntent",
            "slots": [],
            "samples": [
                "go mad",
                "sparkle"
            ]
        }
      ],
      "types": []
    }
  }
}
```


Our skill communicates with our lambda code and vice-versa. They communicate with each other via endpoints. (we saw this in our listener code earlier). Copy the skill endpoint and save it somewhere:

Click **endpoint** on the side-menu and select AWS lambda ARN, e.g:

amzn1.ask.skill.17f5139b-5b80-4abb-9e30-bdc46ef32019

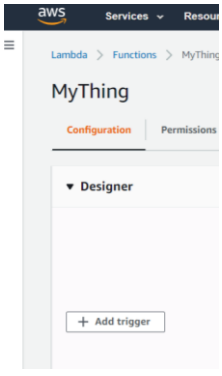
We will fill in the default region with the lambda endpoint saved earlier.



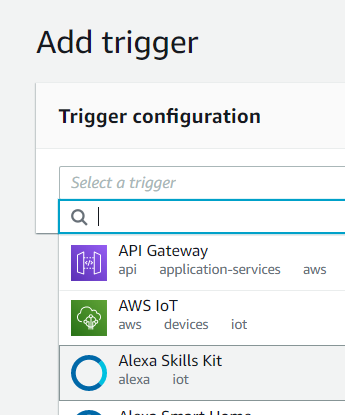

Paste the ARN from the lambda code into the Default region box and then copy the Skill ID.

Click **Save Endpoints**

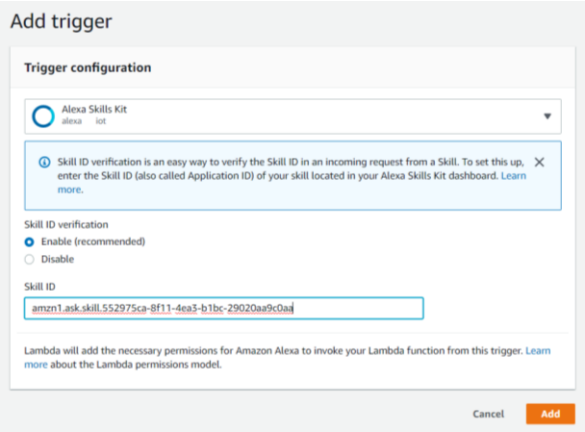
Return to the AWS lambda management console, and click add trigger




Select Alexa Skill skit from the drop-down box:



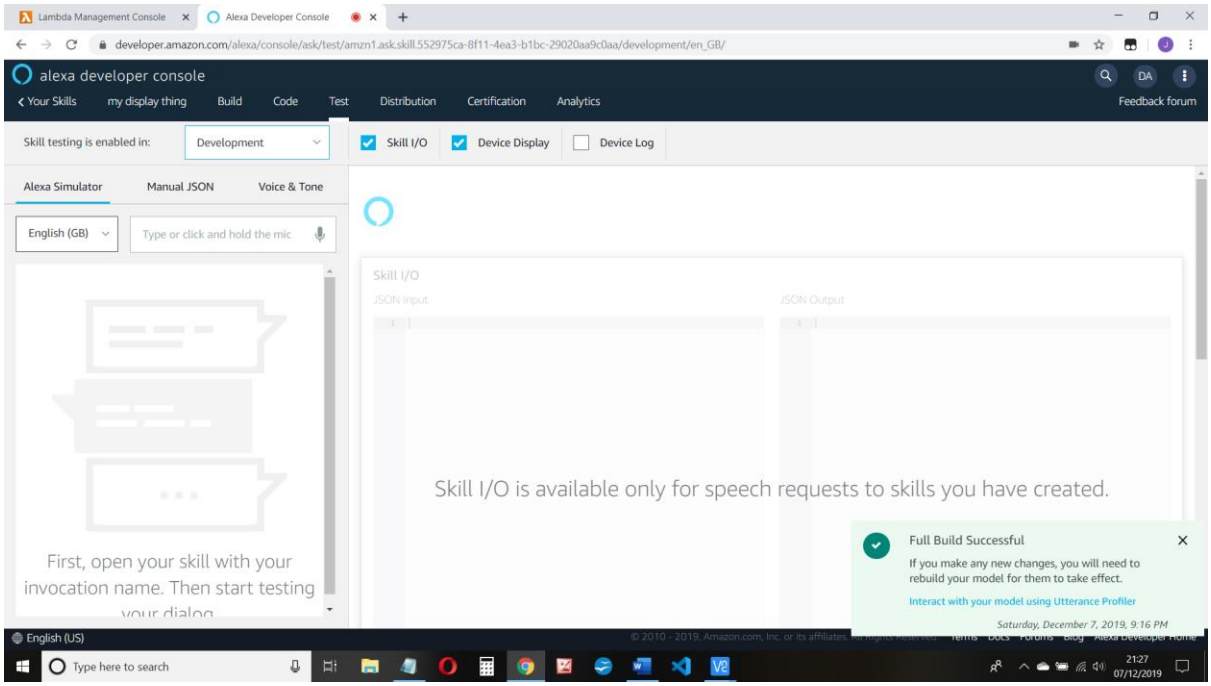

– alexa skills kit paste the skill ID that you copied and click Add




Save everything

## 6. Test the code

Click Test in the Developer followed by the drop-down box from Off to Develop




Don’t forget to start the program on your Pi! Type in the commands (Open my display thing, hello, red etc.). And watch the display.

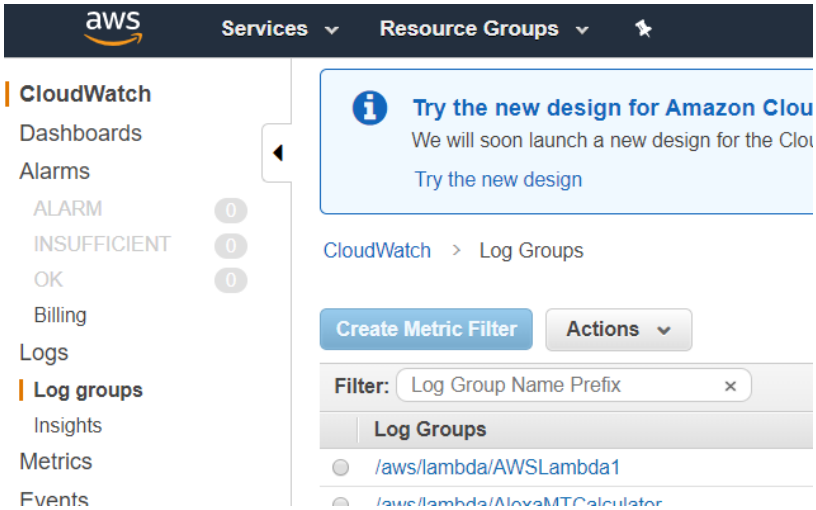
Now try it with your Alexa device!

Well done.

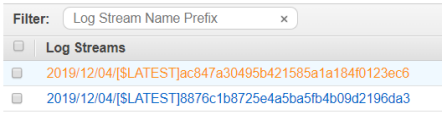
### Cloudwatch logs

It should work, but if not, you can look at the cloudwatch logs

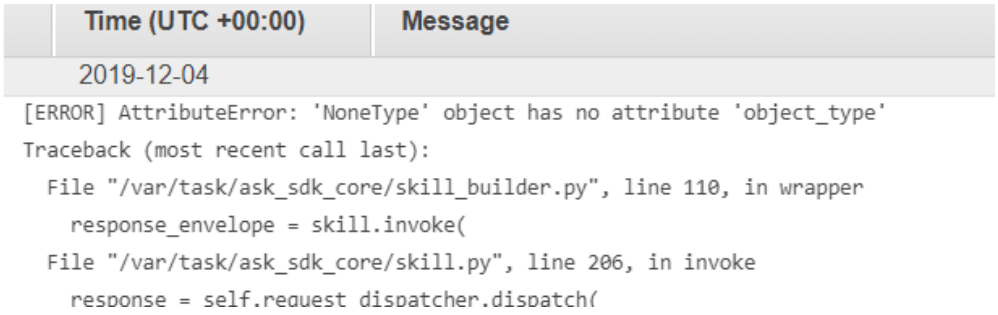
AWS console > CloudWatch  > Log groups




Scroll down to your lambda (e.g. /aws/lambda/MyIoTDisplay) and click the log stream that you want




Scroll through the logs until you find the error:




In this case my code had: send_mqtt_directive("/myPi", hello) – it should be “hello”

To re-upload your code, repeat the commands in section 4.

**How to install RPi OS**

On your PC

Download OS from https://www.raspberrypi.org/downloads/raspbian/

Choose Raspbian Buster with desktop and recommended software ZIP

(currently 2019-07-10-raspbian-buster.zip)

Format your MicroSD card – I used Windows, but you might have to use SD Card formatter if the card is greater than 32M.

Use Balena Etcher to burn the OS.

(whilst it's doing this you can download VNC and WInSCP, Advanced Ip Scanner and Putty)

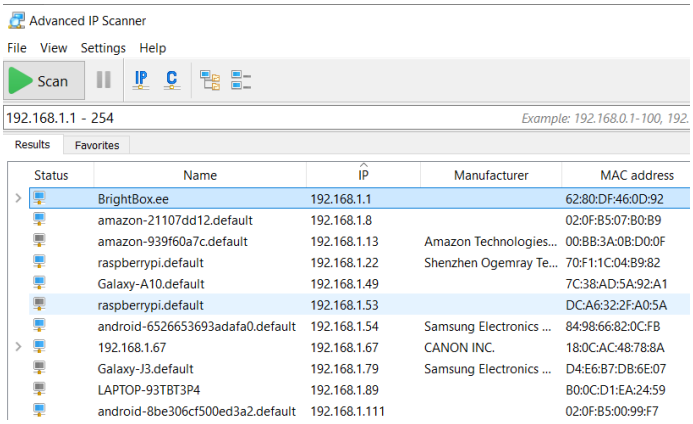
Unplug and re-plug the sd card. Ignore the request to reformat.

Now add ssh file to the boot partition (right-clcik > new > text document > rename to ssh (and remove extension) and also create a wpa_supplicant.conf file which contains the following:

```
country=GB
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
     ssid="your router ID here"
     psk="your router password here"
     key_mgmt=WPA-PSK
   }
```

Eject the SD card, plug it into RPi and power up your RPi

If you can't see your Raspberry Pi using your router, try Advanced IP Scanner.




Here you can see I have two RPis. The new one is 192.168.1.22

Now log in using Putty as Id: pi, password: raspberry

```
sudo raspi-config
```

then set up #5 interfacing options, enable ssh and vnc and #7 advanced options set screen size to max (otherwise VNC may not work)

Update the software: sudo apt-get update

Install any other libraries for your hardware (e.g. sudo apt install sense-hat and then sudo reboot)

Run the colorcycler.py program from

https://github.com/astro-pi/python-sense-hat/blob/master/examples/colour_cycle.py

See also:

https://projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat/4

https://github.com/bennuttall/sense-hat-examples

### Listener.py code

```
#!/usr/bin/python3

import sys
import signal
import time
import json
from time import sleep
from random import randint

from sense_hat import SenseHat
sense = SenseHat()

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

createMQTTClient = AWSIoTMQTTClient("MyThing")
createMQTTClient.configureEndpoint('a3eccx446fyf27-ats.iot.us-east-
1.amazonaws.com', 443)

# Check these certificate names
createMQTTClient.configureCredentials("/home/pi/MyThing/AmazonRootCA1.crt",
"/home/pi/MyThing/MyThing-private.pem.key", "/home/pi/MyThing/MyThing-
certificate.pem.crt")

createMQTTClient.configureOfflinePublishQueueing(-1)  # Infinite queueing
createMQTTClient.configureDrainingFrequency(2)  # Draining: 2 Hz
createMQTTClient.configureConnectDisconnectTimeout(10)  # 10 sec
createMQTTClient.configureMQTTOperationTimeout(5)  # 5 sec

createMQTTClient.connect()
print("Connected")

def unsubscribe_topics():
    """Unbsubscribes from AWS IoT topics before exiting
    """
    print("Unsubscribing")

    topics = [
        '/voice/drive'
    ]

    for topic in topics:
        createMQTTClient.unsubscribe(topic)

# Interrupt Handler useful to break out of the script
def interrupt_handler(signum, frame):
    unsubscribe_topics()
    sys.exit("Exited and unsubscribed")

# Custom MQTT message callbacks
def driveCallback(client, userdata, message):
    print(f"Received {message.payload} from {message.topic}")
    payload = json.loads(message.payload)
    command = payload['directive']
    print(f"Processing command: {command}")
```

```
    if command == "hello":
        sense.show_message("Hi there!")
    elif command == "red":
        sense.clear((255, 0, 0))
    elif command == "sparkle":
        sense.clear((0,0,0))
        timeStart = time.time()
        timeNow = time.time()
        while timeNow-timeStart <5 :
          x = randint(0, 7)
          y = randint(0, 7)
          r = randint(0, 255)
          g = randint(0, 255)
          b = randint(0, 255)
          sense.set_pixel(x, y, r, g, b)
          sleep(0.1)
          timeNow= time.time()
    elif command == "stop":
        sense.show_message("Stopped!")
    else:
        print("Command not found")

# Subscribe to topics
createMQTTClient.subscribe("/voice/drive", 1, driveCallback)
print("Listening on /voice/drive")

while True:
    signal.signal(signal.SIGINT, interrupt_handler)
    time.sleep(1)

unsubscribe_topics()
```

### lambda_function.py code

```
# -*- coding: utf-8 -*-

import random
import logging
import os
import time
import json

from ask_sdk_core.utils import is_intent_name, is_request_type, viewport
from ask_sdk_model.ui import SimpleCard
from ask_sdk_core.skill_builder import SkillBuilder

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

from ask_sdk_model.interfaces.alexa.presentation.apl import (
    RenderDocumentDirective, ExecuteCommandsDirective, SpeakItemCommand,
    AutoPageCommand, HighlightMode)

createMQTTClient = AWSIoTMQTTClient("MyThing")
"""

Change this endpoint to suit yours

"""
createMQTTClient.configureEndpoint('a3egqx446fyf27-ats.iot.us-east-
1.amazonaws.com', 443)
#createMQTTClient.configureEndpoint(os.environ['AWS_IOT_ENDPOINT'], 443)

"""

# Check these certificate names if necessary

"""

createMQTTClient.configureCredentials("./certs/AmazonRootCA1.crt",
"./certs/MyThing-private.pem.key", "./certs/MyThing-certificate.pem.crt")

createMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)
createMQTTClient.configureOfflinePublishQueueing(-1)  # Infinite offline
Publish queueing
createMQTTClient.configureDrainingFrequency(2)  # Draining: 2 Hz
createMQTTClient.configureConnectDisconnectTimeout(10)  # 10 sec
createMQTTClient.configureMQTTOperationTimeout(5)  # 5 sec

createMQTTClient.connect()

SKILL_NAME = "My display controller"
HELP_MESSAGE = "You can say hello, turn the display red, or sparkle."
HELP_REPROMPT = "What do you want to do? Try saying hello?"
STOP_MESSAGE = "Goodbye!"
FALLBACK_MESSAGE = "Oops! I didn't understand. Say hello."
FALLBACK_REPROMPT = 'How can I help you?'
EXCEPTION_MESSAGE = "Sorry. I can't do that"

sb = SkillBuilder()
```

```
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

def _load_apl_document(file_path):
    """Load the apl json document at the path into a dict object."""
    with open(file_path) as f:
        return json.load(f)

def format_mqtt_message(directive, data):
    payload = {}
    payload['directive'] = directive
    payload['data'] = data

    print("Payload")
    print(json.dumps(payload))

    return json.dumps(payload)

def send_mqtt_directive(topic, directive, data = {}):
    payload = format_mqtt_message(directive, data)
    createMQTTClient.publish(topic, payload, 1)

@sb.request_handler(can_handle_func = is_intent_name("SparkleIntent"))
def spin_around_intent_handler(handler_input):
    speech = "Ok, sparkling"
    send_mqtt_directive("/myPi", "sparkle")
# send_mqtt_directive("/myPi", "spin", data= {"data", 10))
# replace 10 with passed sparkle time


handler_input.response_builder.speak(speech).set_card(SimpleCard(SKILL_NAME
, speech)).set_should_end_session(False)
    return handler_input.response_builder.response

@sb.request_handler(can_handle_func = is_intent_name("RedIntent"))
def spin_around_intent_handler(handler_input):
    speech = "Ok, turning red"
    send_mqtt_directive("/myPi", "red")


handler_input.response_builder.speak(speech).set_card(SimpleCard(SKILL_NAME
, speech)).set_should_end_session(False)
    return handler_input.response_builder.response

@sb.request_handler(can_handle_func = is_intent_name("StopIntent"))
def stop_moving_intent_handler(handler_input):
    speech = "Ok, stopping"
    send_mqtt_directive("/myPi", "stop")


handler_input.response_builder.speak(speech).set_card(SimpleCard(SKILL_NAME
, speech)).set_should_end_session(False)
    return handler_input.response_builder.response

@sb.request_handler(can_handle_func=is_request_type("LaunchRequest"))
def launch_request_handler(handler_input):
```

```
    speech = "Hi! You can control the display, say hello, or give me a
command like sparkle or turn red"


handler_input.response_builder.speak(speech).set_card(SimpleCard(SKILL_NAME
, speech)).set_should_end_session(False)
    return handler_input.response_builder.response

@sb.request_handler(can_handle_func = is_intent_name("HelloWorldIntent"))
def hello_world_intent_handler(handler_input):

    speech = "Hello. Look at the display!"
    send_mqtt_directive("/myPi", "hello")
    print("Send hello to MQTT")


handler_input.response_builder.speak(speech).set_card(SimpleCard(SKILL_NAME
, speech)).set_should_end_session(False)
    return handler_input.response_builder.response

@sb.request_handler(can_handle_func = is_intent_name("AMAZON.StopIntent"))
def help_intent_hanlder(handler_input):
    speech = "Ok, stopping"
    send_mqtt_directive("/myPi", "stop")
    return
handler_input.response_builder.speak(HELP_MESSAGE).ask(HELP_REPROMPT).set_c
ard(SimpleCard(SKILL_NAME, HELP_MESSAGE)).response

@sb.request_handler(can_handle_func = is_intent_name("AMAZON.HelpIntent"))
def help_intent_hanlder(handler_input):
    return
handler_input.response_builder.speak(HELP_MESSAGE).ask(HELP_REPROMPT).set_c
ard(SimpleCard(SKILL_NAME, HELP_MESSAGE)).response

@sb.request_handler(can_handle_func =
(is_intent_name("AMAZON.CancelIntent") or
is_intent_name("AMAZON.StopIntent")))
def cancel_or_stop_intent_handler(handler_input):
    return handler_input.response_builder.speak(STOP_MESSAGE).response

@sb.request_handler(can_handle_func =
is_intent_name("AMAZON.FallbackIntent"))
def fallback_intent_handler(handler_input):
    return
handler_input.response_builder.speak(FALLBACK_MESSAGE).ask(FALLBACK_REPROMP
T).set_card(SimpleCard(SKILL_NAME, FALLBACK_MESSAGE)).response

@sb.request_handler(can_handle_func =
is_request_type("SessionEndedRequest"))
def session_ended_request(handler_input):
    logger.info("In SessionEndedRequestHandler")
    logger.info("Session ended reason:
{}".format(handler_input.request_envelope.request.reason))
    return handler_input.response_builder.response

@sb.exception_handler(can_handle_func = lambda i, e: 'AskSdk' in
e.__class__.__name__)
```

```
def ask_exception_intent_handler(handler_input, exception):
    return
handler_input.response_builder.speak(EXCEPTION_MESSAGE).ask(HELP_REPROMPT).
response

@sb.global_request_interceptor()
def request_logger(handler_input):
    print("Request received:
{}".format(handler_input.request_envelope.request))

# Handler name that is used on AWS lambda
lambda_handler = sb.lambda_handler()
```

**Quick instruction list:**

Sign in to the AWS IoT Console

Register a Device in the Registry

Configure Your Device

View Device MQTT Messages with the AWS IoT MQTT Client

Sign in to the AWS IoT Console, register a device

Manage > Register a Thing  or

Manage > Create

Register a single AWS IoT thing > Create a single thing

Name > MyThing > Next

One-click certificate creation > Create certificate

Download the certs. (and keep cert.pem)

Choose Download for the Amazon root CA. > Choose RSA 2048 bit key: Amazon Root CA 1 > Choose CA Certificates for Service Authentication > Choose Amazon Root CA 1 .

Copy this text and paste it into a file named **AmazonRootCA1.crt**.

Return to the cerificates page and click **Activate**

Now we will **attach a policy**. Click that.

First click Register a Thing.

Now we will create a policy for out ‘thing’

Go to the IoT main page, click Secure > Policies > Create a Policy

Give it a name: MyThingPolicy

Add an Action: enter **iot:***. And for Resource ARN, enter ***.

Under Effect, choose **Allow**, and then choose **Create**.

Find your endpoint.

On your Pi, create the listener.py code, edit the certs and your endpoint. Run the code

On AWS IoT > Test, add a drive called /myPi and publish commands to it to test the listener code.

On your Pi, create the lambda_function.py code, copy the certs to the lambda/certs folder, edit the code for the certs and the endpoint.

Bundle up the dependencies and certs and lambda_function.py code.

On AWS create a new lambda function and upload the code. Remember the endpoint. Add an Alexa Skill trigger.

On the Alexa Developer console, create a new Skill, add the utterances and intents, (or copy the JSON). Go to endpoint and add the ARN from the lambda. Copy the SkillID and paste that into the Lambda trigger.

Test the code from the Developer page. Finally test it with your Alexa Device.

#4 Create the Alexa skill and link to the lambda program