# Chapter 7b: Cloud Connectivity using MQTT + Amazon Web Services

## Objective

At this end of Chapter 7b you will understand:

* How Message Queue Telemetry Transport (MQTT) works:
  + How MQTT fits into the TCP/IP Networking Stack
  + The purpose of a **Topic**
  + The role of the **Message Broker**
  + The role of a **Publisher**
  + The role of a **Subscriber**
  + The definition of MQTT Quality of Service (QOS)
  + What are the Ports used by MQTT
  + How the Keep Alive functions.
* How to use the JavaScript Object Notation (JSON) language.
* How the Amazon AWS MQTT Cloud works including:
  + Know how to provision “things” in the Amazon Cloud by creating things, policies and certificates
  + AWS MQTT Security
  + Understand “shadow device”
  + Know how to use an AWS MQTT Client to subscribe and publish to topics
  + Understand the scope of systems that can be implemented in the AWS Cloud (SNS, Database etc.)
  + Understand the role of public and private key cryptography in the AWS cloud
* Understand in **DETAIL** how to write WICED firmware to interact with the AWS IOT Cloud

## Time: 4 Hours

## Fundamentals

### (Application Protocol) MQQT

#### Topic

#### Publisher

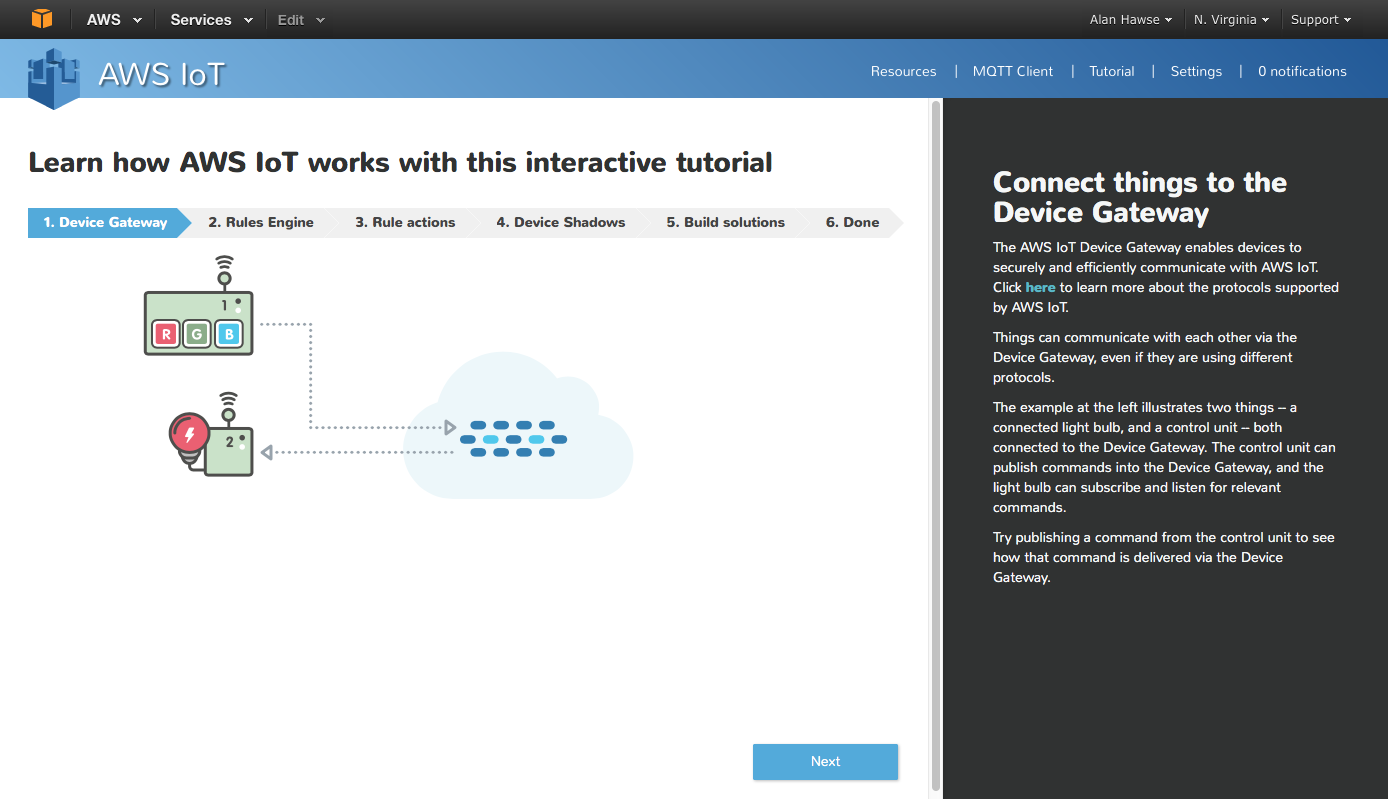
#### Subscriber

### (Data Format) JSON

### (Cloud) AWS IoT

## Exercise(s)

### 00 Run the MQTT tutorial on Amazon IoT Console



### 01 Provision a new thing in the AWS IOT Cloud, establish its policy and credentials and test using the AWS MQTT Client

To Provision follow the “Procedure to Provision an AWS IoT Thing” section at the end of this chapter

To test your Message Broker, follow the “Procedure to use the AWS MQTT Client” at the end of this chapter.

### 02 Build and test the demo.aws\_iot.pub\_sub.publisher App

1. Copy the apps/demo/aws\_iot/publisher to your directory and fix all of the files
2. Modify the DCT for your network
3. Copy your certificates into the resources/apps/aws\_iot directory. You need to name your device certificate to be “client.cer”, your private key to be “privkey.cer”
4. Create the Make target for your project
5. Modify the #defines for WICED\_TOPIC, MQTT\_BROKER\_ADDRESS
6. Build and program your project
7. Watch your terminal session
8. Subscribe to the topic using the AWS MQTT client (and test the publishing)

### 03 Explain in detail the firmware flow for publisher app by answering the following questions:

How do the MQTT library functions (e.g. wiced\_mqtt\_publish ) get into your project?

What function is called when the button is pressed?

How does the button callback unlock the main thread?

What WICED-SDK RTOS mechanism does the “wait\_for\_result” function use to “wait”?

Why did the firmware author create a function called “wait\_for\_result”?

Are all message sent to the AWS IOT MQTT Message broker required to be in JSON format?

What are the 7 WICED MQTT events? What file are they defined in?

Do you have to name the client certificate client.cer? How would you change the name?

What is the naming convention used to differentiate WICED MQTT library functions versus wrappers around those functions in the publisher app?

What is the sequence of function calls to open a connection?

What prevents a hung connection from deadlocking the publisher app?

What is the name of the flag that prevents the firmware from sending multiple button presses before the publish is finished?

To open a connection, the flow looks like this:

* mqtt\_conn\_open calls wiced\_mqtt\_connect with a call back of mqtt\_connection\_event\_cb
* mqtt\_conn\_open calls wait\_for\_result which gets the msg\_semaphore (which halts everything)
* When Amazon responds, the mqtt\_connecton\_event\_cb occurs and wakes up the thread
* The mqtt\_connection\_event\_cb updates the “expected\_event” global variable and sets the msg\_semaphore then returns
* Back in wait\_for\_event If the message WICED\_MQTT\_EVENT\_TYPE\_CONNECT\_REQ\_STATUS is received then it then it returns WICED\_SUCCESS else it returns WICED\_FAIL

What is the equivalent flow to publish?

### 04 Modify (02) by adding debugging prints to the function mqtt\_connection\_event\_cb

1. Watch the events as each of the events occurs during the MQTT process

### 05 Implement the app.aws\_iot.pub\_sub.subscriber App

1. Copy the app to your directory and fixup the DCT and Makefile
2. Update the topic and broker #defines to the same one you chose for (03)
3. Publish messages using the AWS MQTT Client

### 06 Implement the subscriber and publisher in two different kits and test

1. (not required) Make another AWS thing

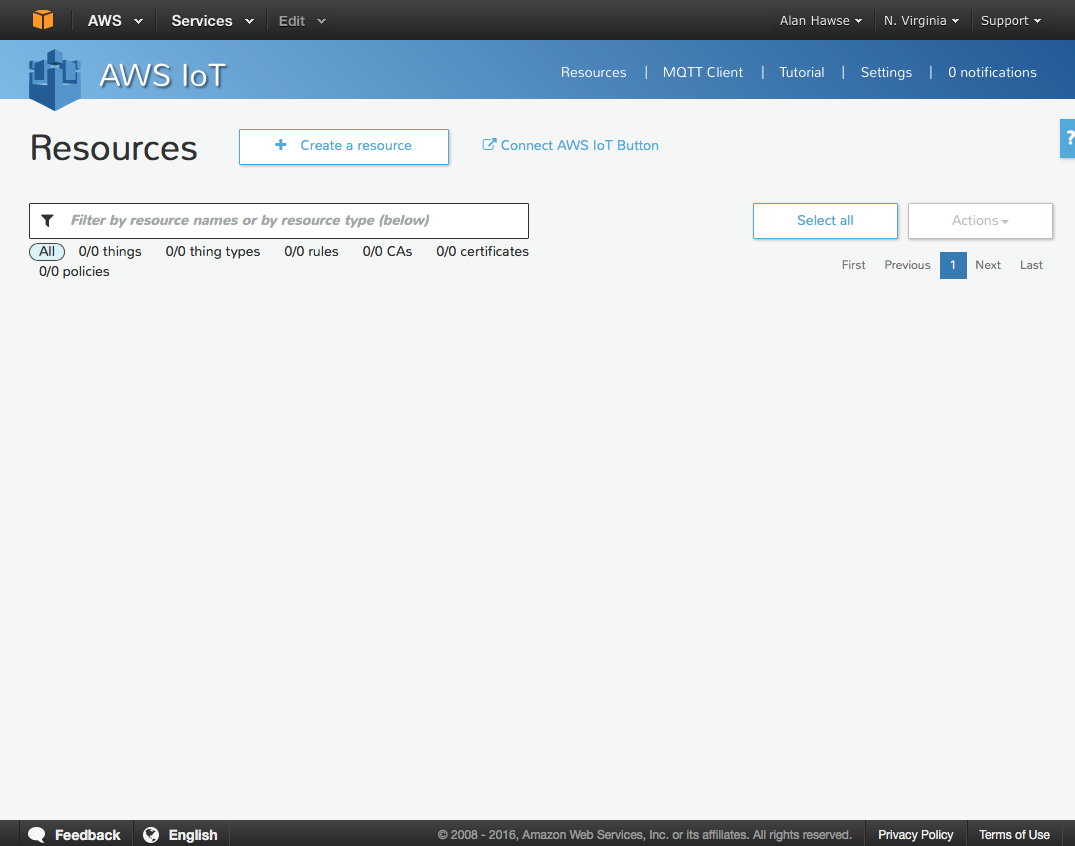
### 07 Implement the Shadow application

1. Copy the app to your directory and update Makefile
2. Update the DCT to have a SoftAP name that is unqiue
3. Fix the message broker address and the MQTT Topic
4. Attach to the App using the Soft AP on your board
   1. Connect to the SSID that you programmed into the board
   2. Go to the webserver (The IP address is printed on the terminal when the device boots and starts the AP)
   3. Follow the instruction to setup the station SSID
   4. Follow the instructions to upload the keys
   5. The board will reboot. Once it has done that it will attach to the network as a Station
5. Start the MQTT Client on AWS
6. Press the button on the board and see the message
7. Answer the question “What is the sequence of events that changes the LED from On to Off?”

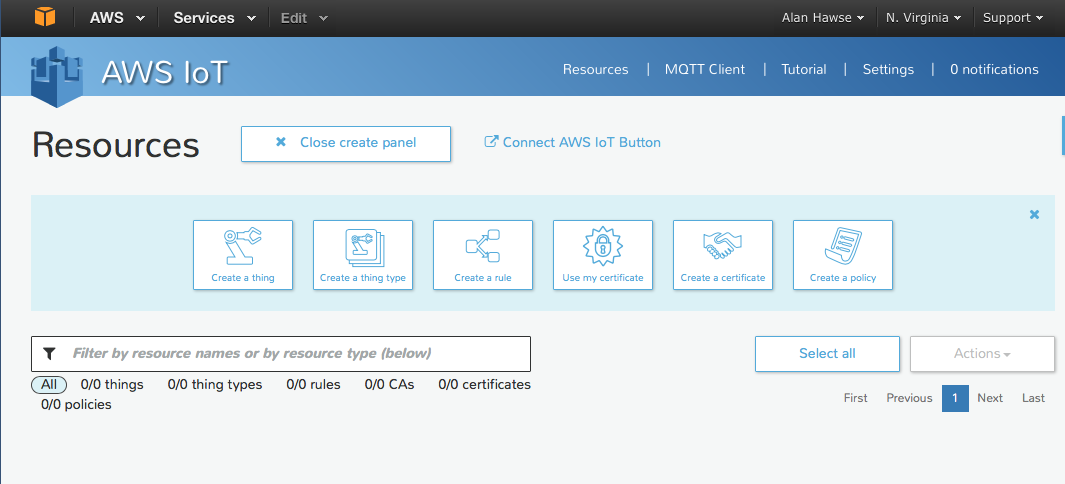
## Related Example “Apps”

## Procedure to Provision an AWS IoT Thing

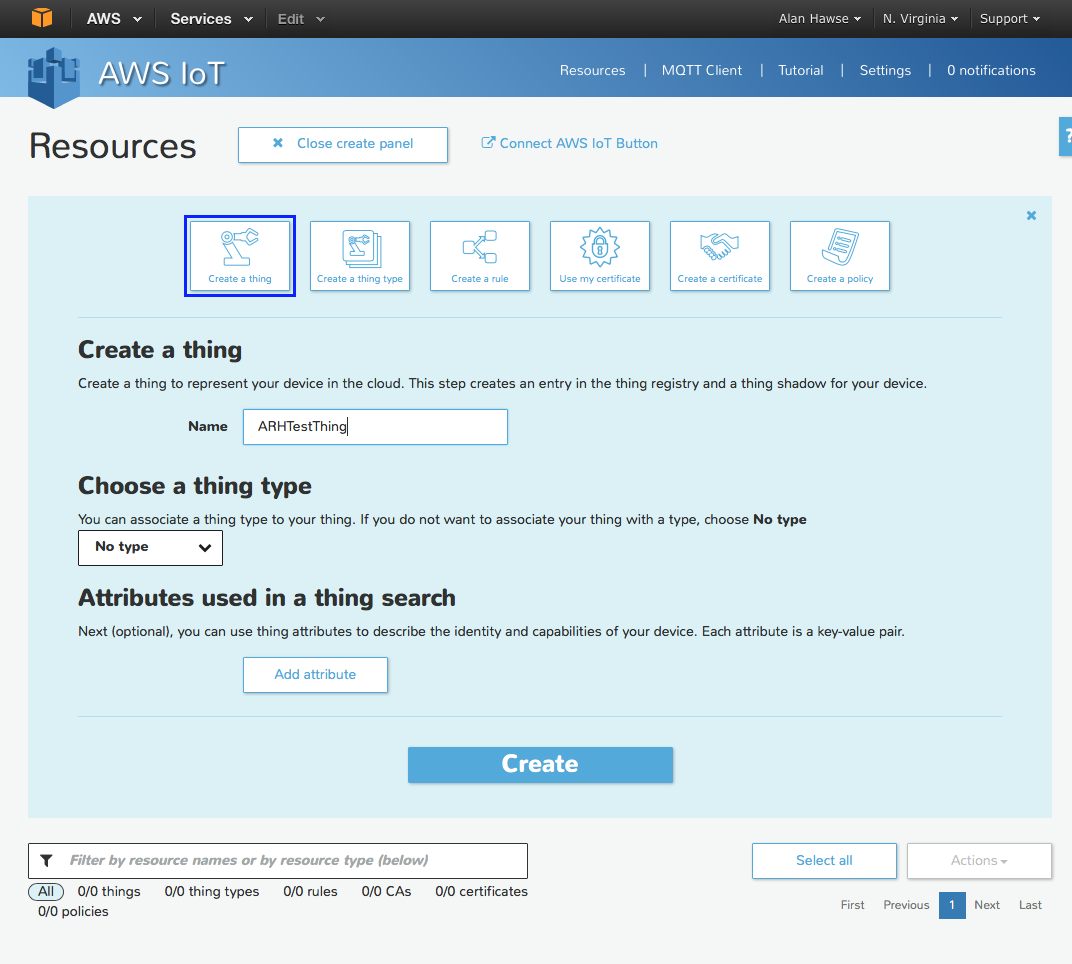
1. Start by going to the AWS IoT Console which will start as blank



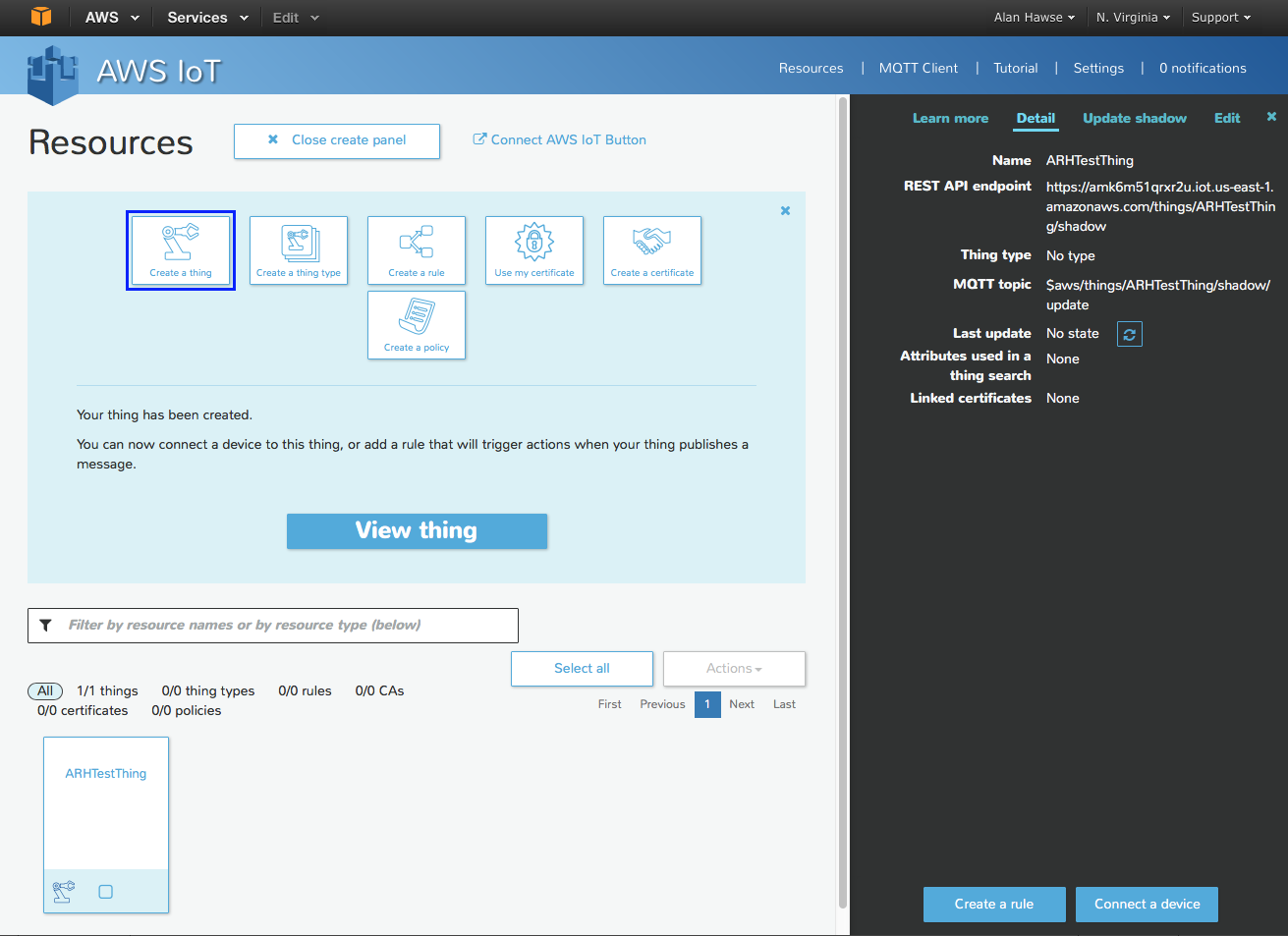
1. Press “Create a thing”



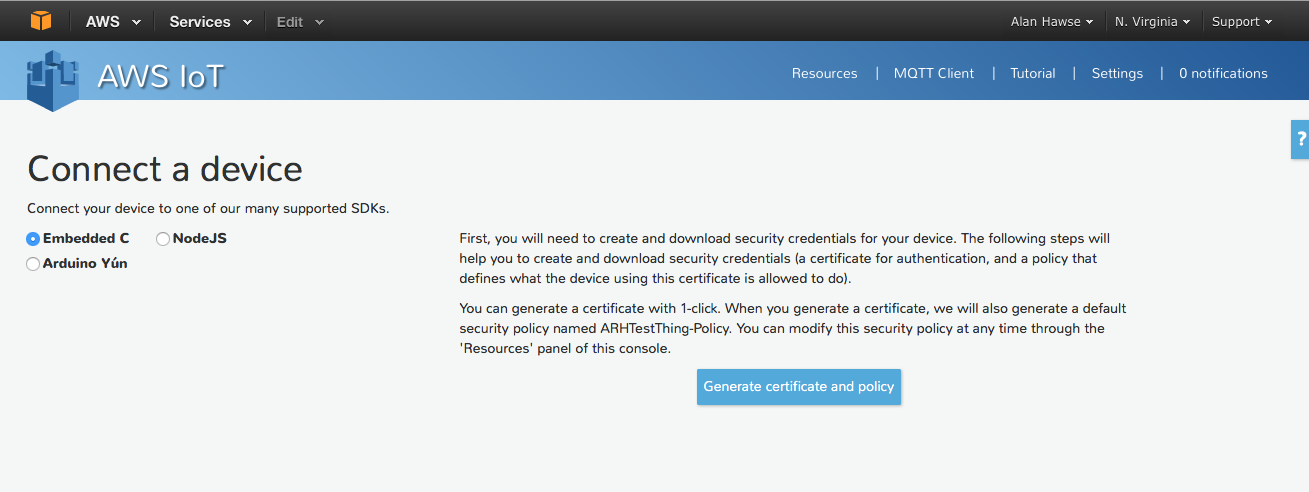
1. Name it “ARHTestThing” (or whatever) and press “Create”



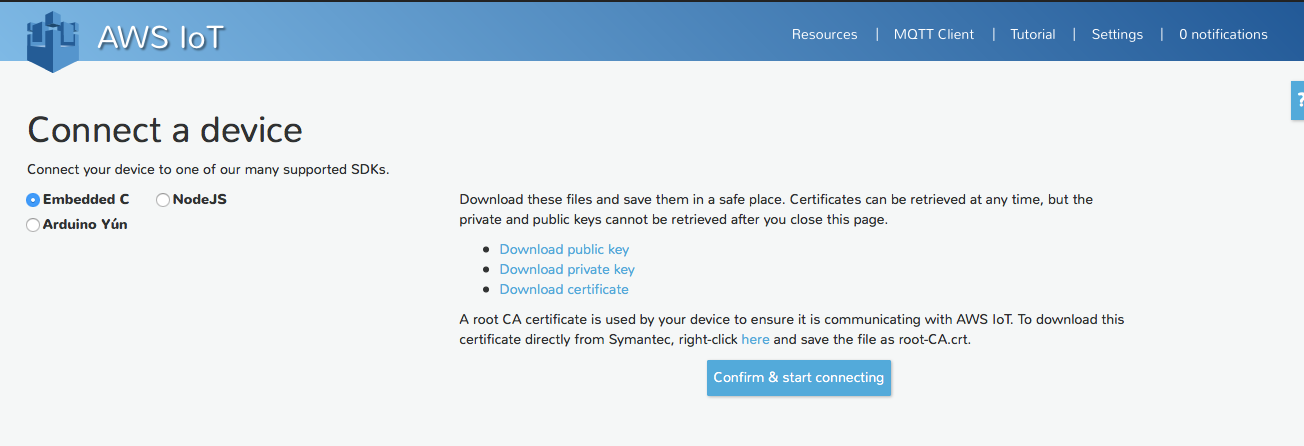
1. Click on your “thing” and it will open a window up to the right. Before you can do anything with the “thing” you need to create the encryption keys that enable you to identify yourself as that thing, and then update the data. To do this, click on “Connect Device”



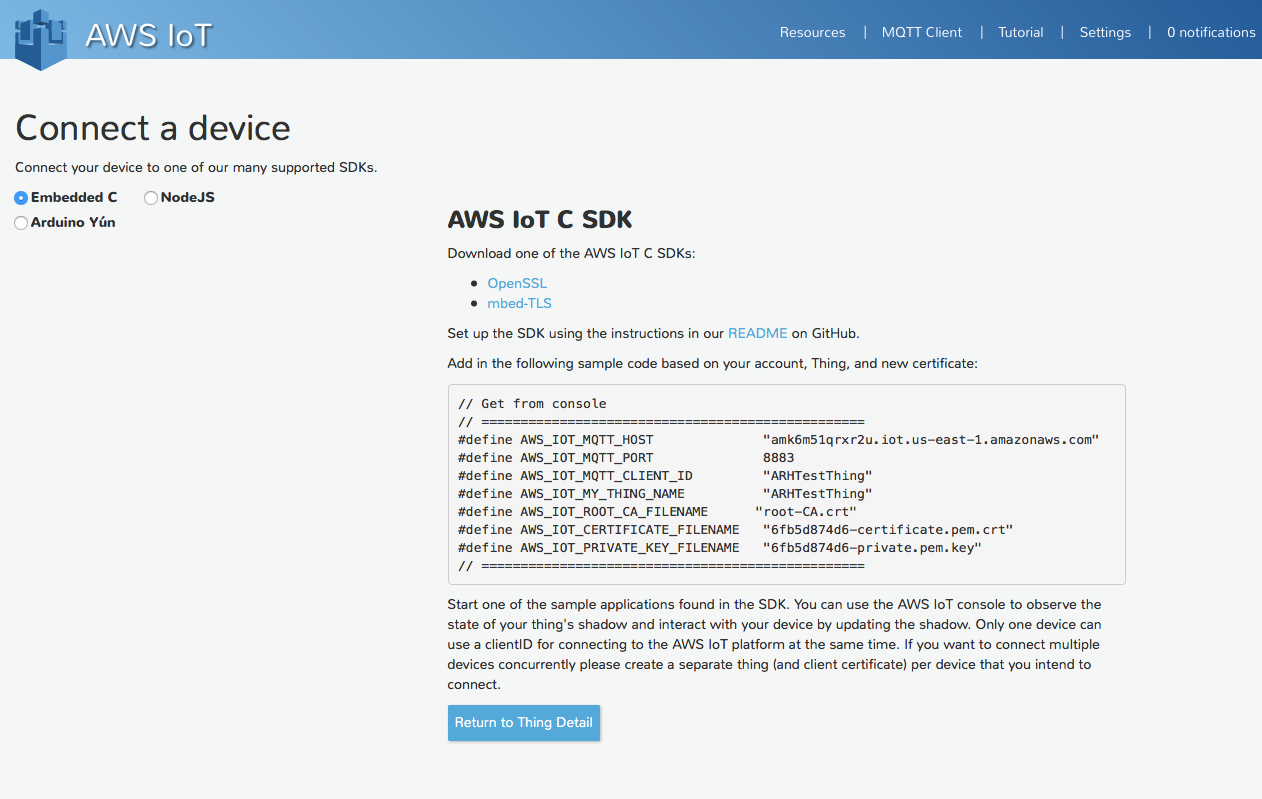
1. Pick “Embedded C” and then click “Generate certificate and policy”



1. Now you need to save the “public key”, “private key” and “certificate”. If you forget this step you cannot come back… so really you must have those files to make the TLS work

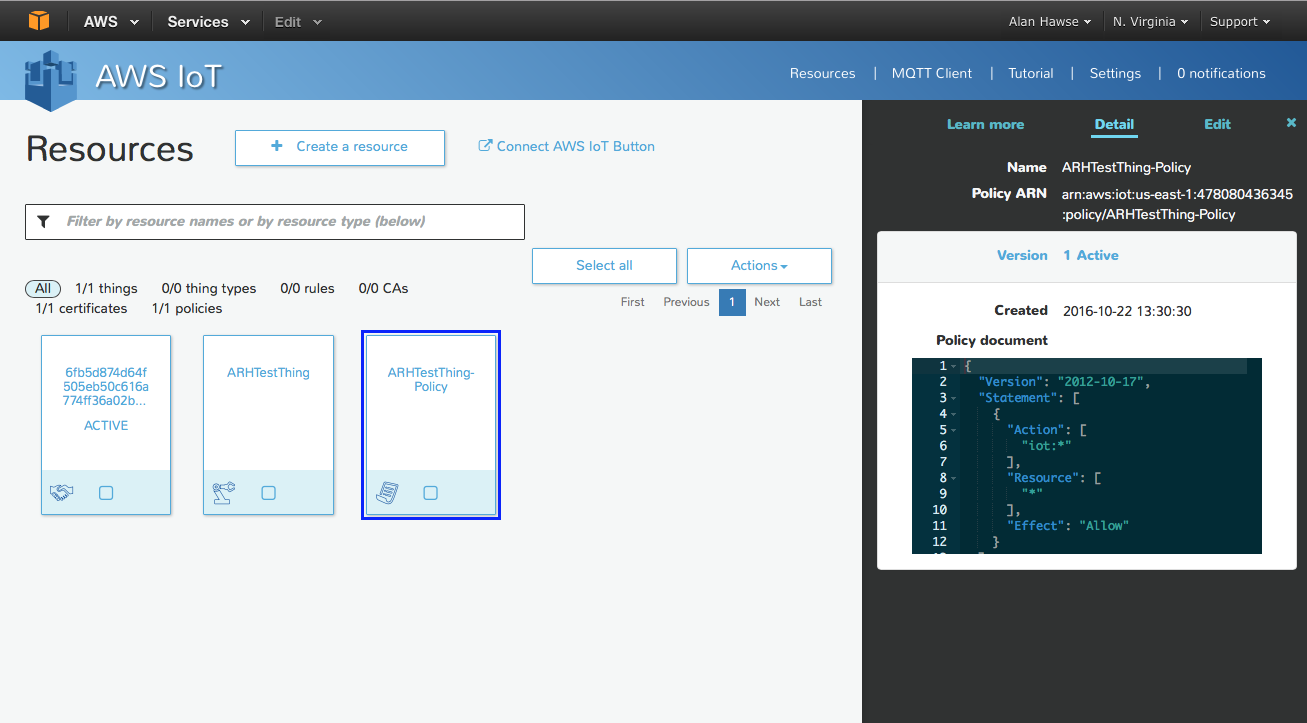


1. Save the embedded C (copy/paste) then press “Return to Thing Detail”



1. Click on the Certificate. What this tells AWS is that when you are identified with this certificate that you are “ARHTestThing” and that you are allow to follow the policy “ARHTestThing-Policy”

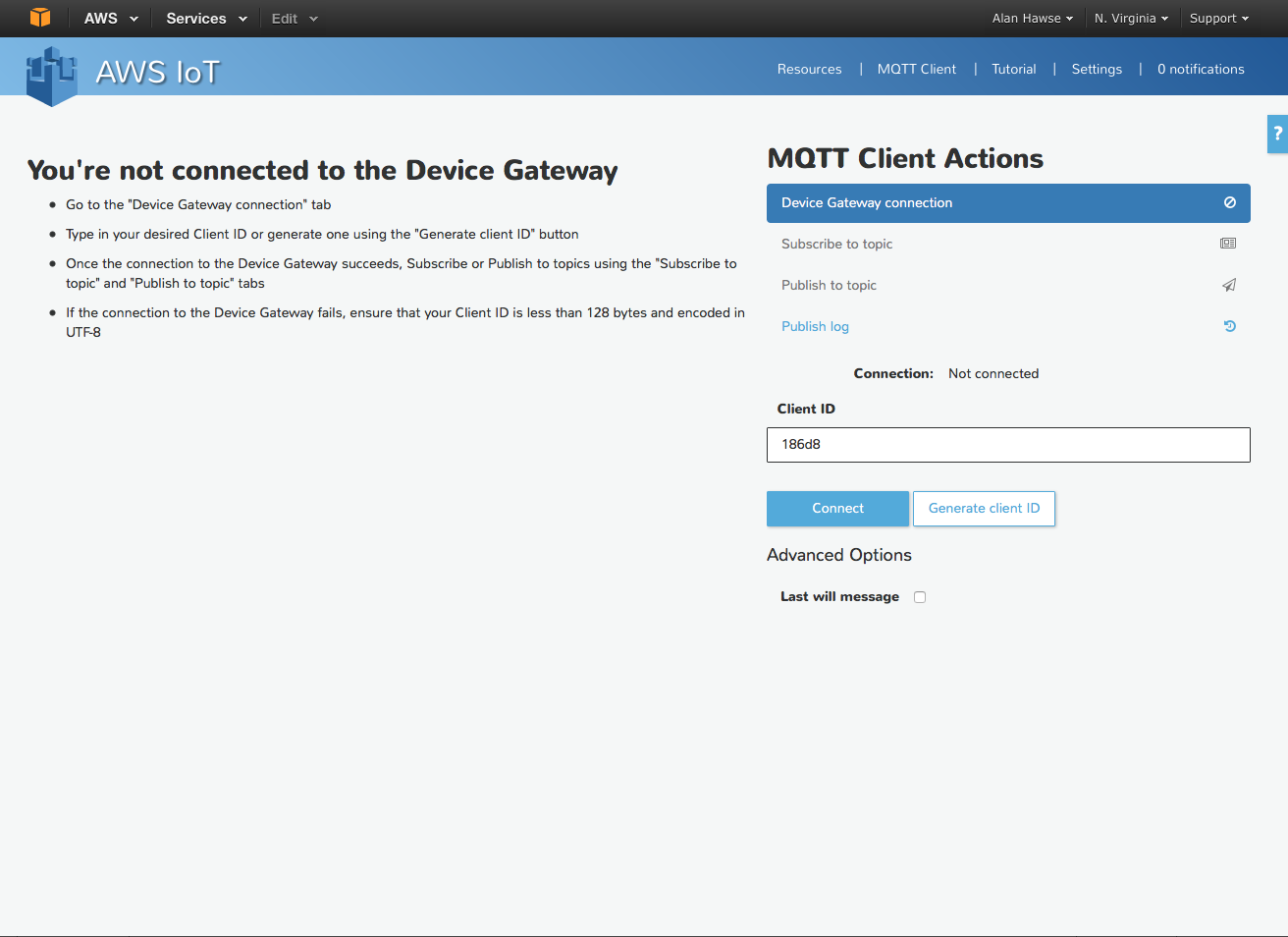


1. Click on your “things” policy

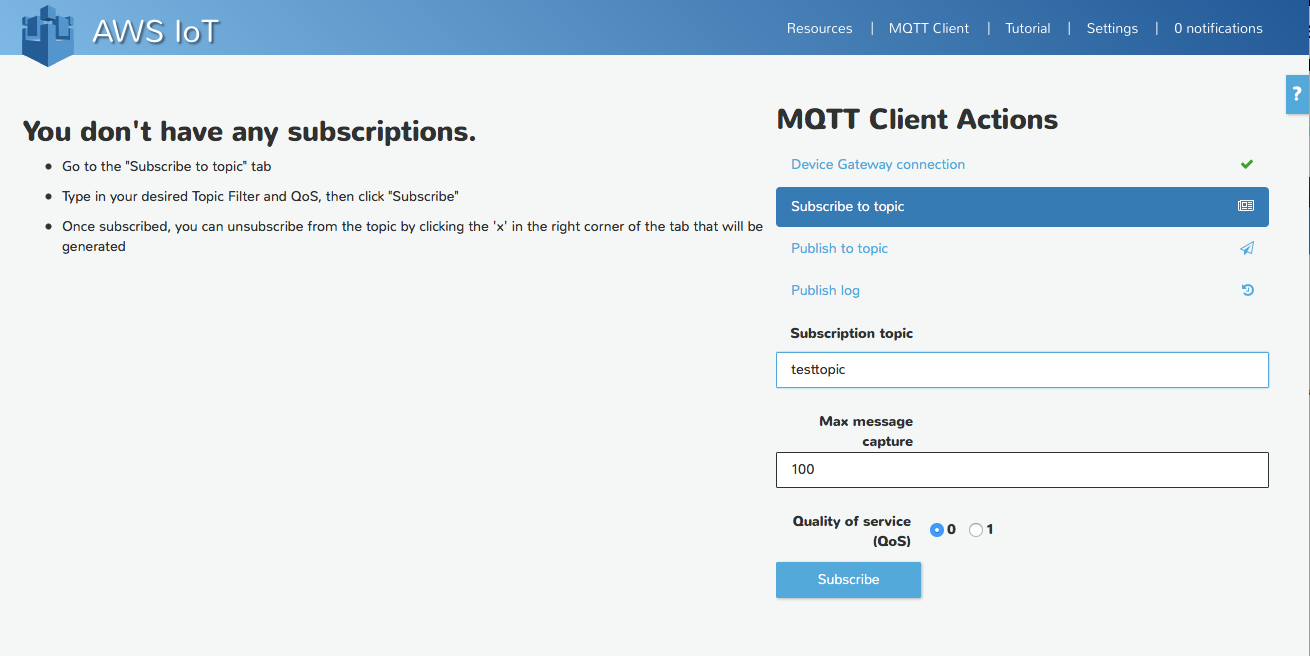
## Procedure to use the AWS MQTT Client

The AWS MQTT Client is a Web Browser based client that you can connect to “your” message broker. Then you can publish and subscribe to topics. You can think of it as an IoT thing that can publish and subscribe. To use it to test you can:

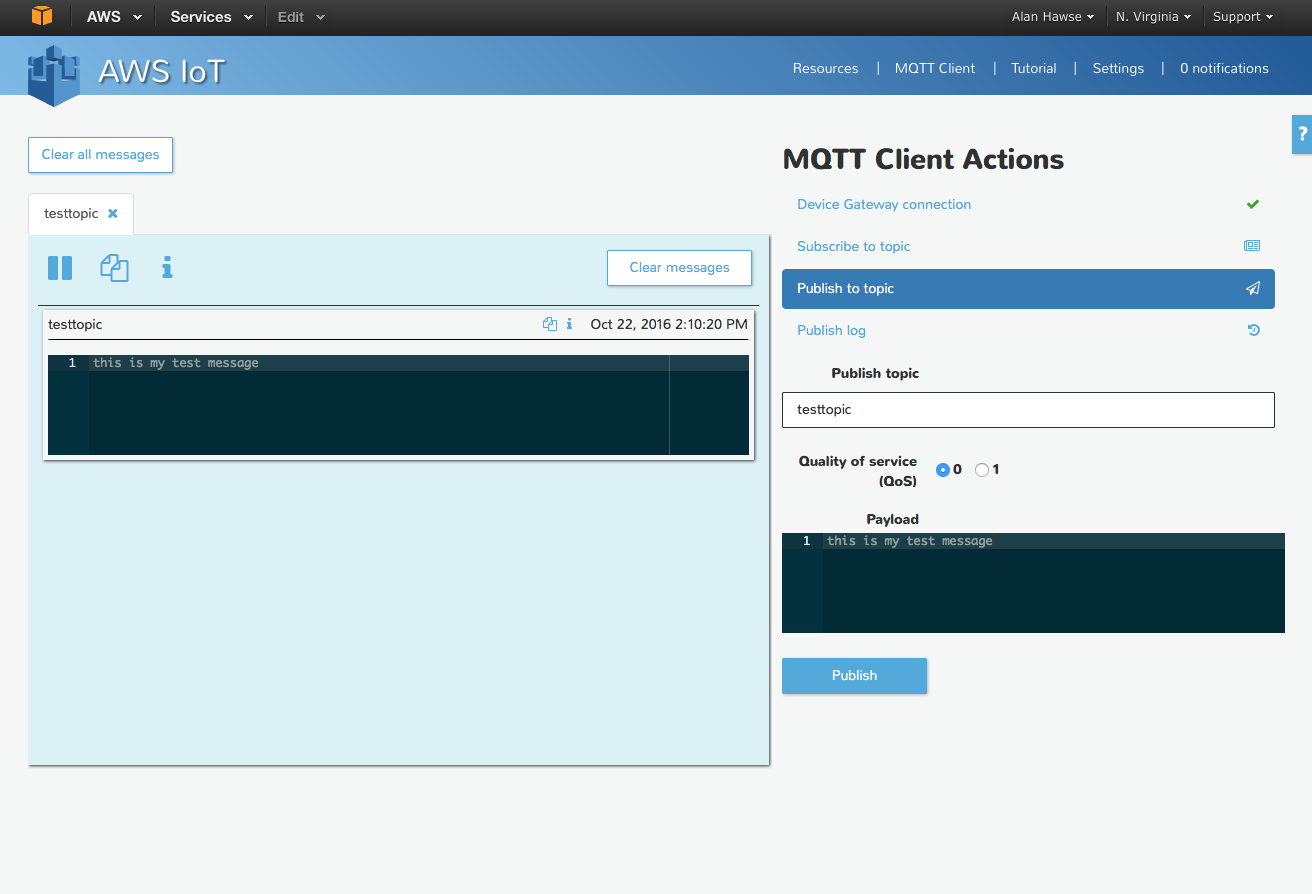
1. Click on “MQTT Client”. Then give your client a unique name (like ARHTestThing from the previous example but don’t use that name as it is already assigned). You can click “Generate client ID” and AWS will think up a random-ish name (which is what I did in the screen shot below). Once you have a name click “Connect” which will hook you up to the message broker.



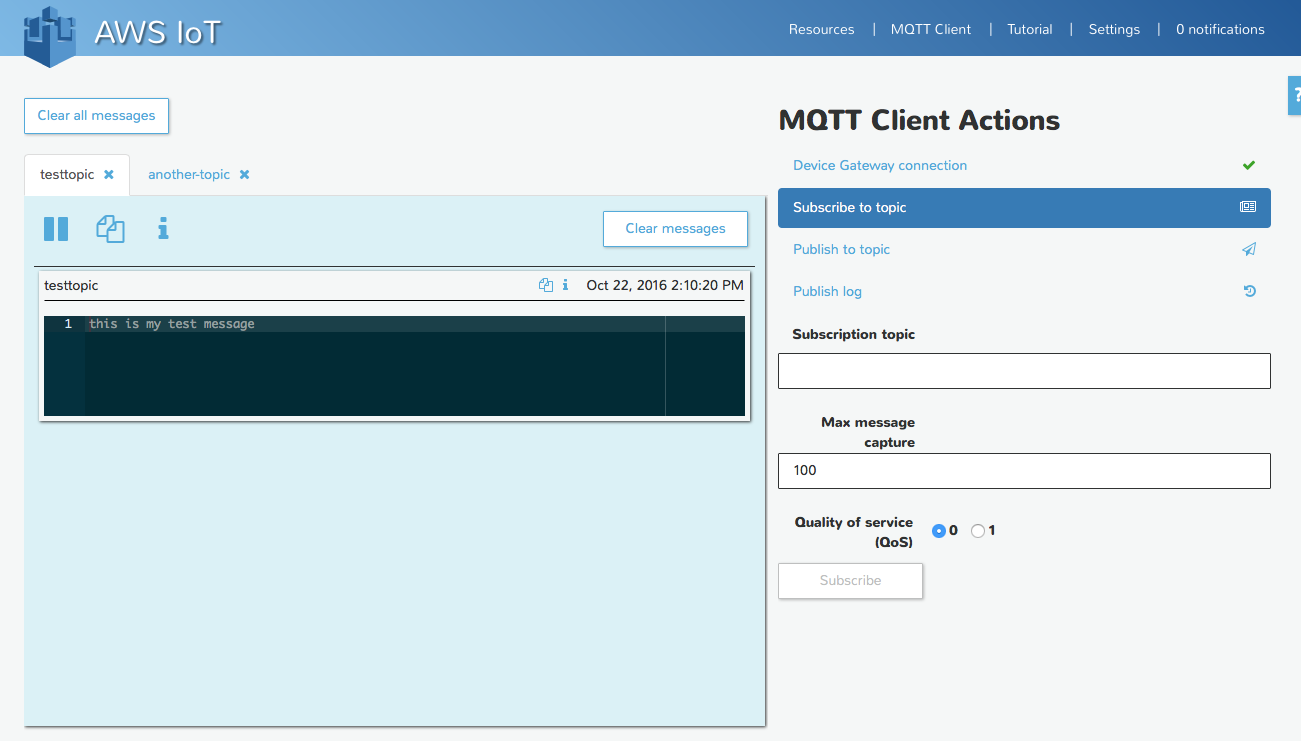
1. Now that you have a connection, the first thing to do is Subscribe to a topic. Remember that there is nothing particularly special about the topic name. Lets choose “testtopic”



1. Now that I am subscribed to a topic I can now send messages to that topic. To do this press “Publish to topic”. Then fill in the name of the topic, in the previous step we called the topic “testtopic”. Then type in your “Payload” and press “Publish”. You can see in the box below I sent the message “this is my test message”.



1. If I were to publish to the topic “testtopic” from my IoT device I would see the message appear in the window on the left side of the screen. The message viewer is cool because you can subscribe to multiple topics at the same time. You can see I did just that by subscribing to “another-topic”



1. In the screen shot below you can see that I create a WICED App that connected to the Message Broker and published “LIGHT ON” to “testtopic”

