# 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**
* How to use the JavaScript Object Notation (JSON) language.
* How the Amazon [AWS MQTT Cloud works](https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html) including:
  + How to provision “things” (which for semantic reasons, will be notated *thing***,** a.k.a. your IoT device, in this chapter) in the Amazon AWS IoT Cloud by creating *thing*, policies and certificates.
  + AWS MQTT and Security
  + Understand the *thing* shadow
  + How to use the AWS IoT 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 in **DETAIL** how to write WICED firmware to interact with the AWS IOT Cloud

## Time: 4 Hours

## Fundamentals

### Amazon Web Services (AWS)

[AWS](https://aws.amazon.com/what-is-aws/) is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality (which makes more money for Amazon than their retail operations). AWS is built from a vast array of both virtual and actual servers and networks as well as a boatload of webserver software and administrative tools including (partial list):

* [AWS IoT](https://aws.amazon.com/iot/): A cloud platform that provides Cloud services for IoT devices (the subject of this chapter).
* Amazon Elastic Cloud ([EC2](https://aws.amazon.com/ec2/)/[EC3](https://aws.amazon.com/ec3/)): A virtualized compute capability, basically Linux, Windows etc. servers that you can rent.
* [Amazon Lambda](https://aws.amazon.com/lambda/): A Cloud service that enables you to send event driven tasks to be executed.
* Storage: Large fast file systems called [Amazon S3](https://aws.amazon.com/s3/?sc_channel=PS&sc_campaign=acquisition_US&sc_publisher=google&sc_medium=s3_b&sc_content=s3_e&sc_detail=amazon%20s3&sc_category=s3&sc_segment=105589469922&sc_matchtype=e&sc_country=US&s_kwcid=AL!4422!3!105589469922!e!!g!!a) & [AWS Elastic File System](https://aws.amazon.com/efs/).
* Databases: Large fast databases called [Dynamo DB](https://aws.amazon.com/dynamodb/), [Amazon Relational Database (RDS)](https://aws.amazon.com/rds/?sc_channel=PS&sc_campaign=acquisition_US&sc_publisher=google&sc_medium=rds_b&sc_content=rds_e&sc_detail=amazon%20rds&sc_category=rds&sc_segment=145495376109&sc_matchtype=e&sc_country=US&s_kwcid=AL!4422!3!145495376109!e!), [Amazon Aurora](https://aws.amazon.com/rds/aurora/?sc_channel=PS&sc_campaign=acquisition_US&sc_publisher=google&sc_medium=aurora_rds_b&sc_content=aurora_e&sc_detail=amazon%20aurora&sc_category=aurora&sc_segment=145509396189&sc_matchtype=e&sc_country=US&s_kwcid=AL).
* Networking: Fast, fault tolerant, load balanced networks with entry points all over the world.
* Developer tools: A unified programming API supporting the AWS platform supporting a bunch of different languages.
* [Amazon Simple Notification System (SNS)](https://aws.amazon.com/sns/): A platform to send messages including SMS and Email.
* [Amazon Simple Queueing Services(SQS)](https://aws.amazon.com/sqs/): A platform to send messages between servers (NOT the same thing a MQTT messages).
* [Amazon Kinesis](https://aws.amazon.com/kinesis/): A platform to stream and analyze “massive” amounts of data. This is the plumbing for AWS IoT.

### Amazon AWS IoT Introduction

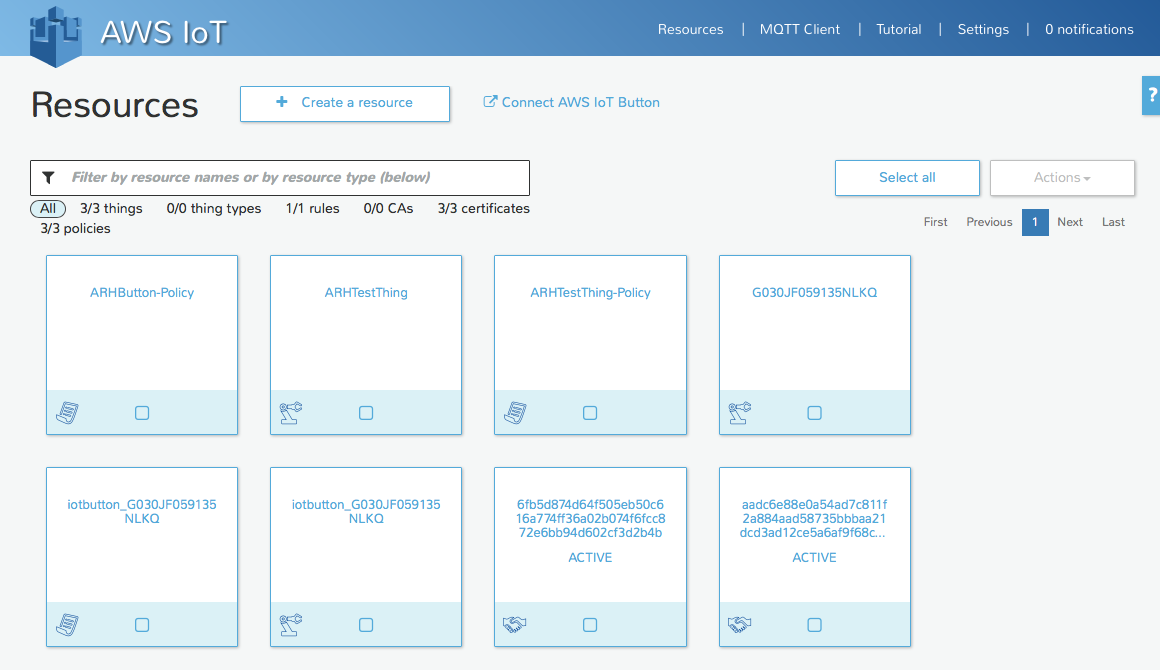
The Amazon AWS IoT Cloud service is an MQTT Message Broker **plus** a bunch of server side functionality that provides:

* Message Broker: An MQTT Message Broker.
* Thing Registry: A web interface to manage the access to your *things*.
* Security and identity: A web interface to manage the certificates and rules about *things*. You can create encryption keys and manage access privileges.
* A “shadow”: An online cache of the most recent state of your *thing***.**
* Rules Engine: An application that runs in the cloud that can subscribe to Topics and take programmatic actions based on messages – for example, you could configure it to subscribe to an “Alert” topic, and if a *thing* publishes a warning message to the alert topic, it uses Amazon SNS to send a SMS Text Message to your cell phone
* IoT Applications: An SDK to build Web pages and cell phone Apps.



### Amazon AWS IoT Resources

When you enter the [AWS IoT Console](https://console.aws.amazon.com/) (console.aws.amazon.com) you will see the Registry which shows all of the resources you have defined. There are three types of resources: *Things*, Certificates, and Policies. The second exercise will take you step by step through the process to create each of them.



**Thing**

A *thing* is a representation of a device or logical entity. It can be a physical device or sensor (for example, a light bulb or a switch on a wall). It can also be a logical entity like an instance of an application or a physical entity that does not connect to AWS IoT but can be related to other devices that do (for example, a car that has engine sensors or a control panel).

**Certificate**

AWS IoT provides mutual authentication and encryption at all points of connection so that data is never exchanged between *things* and AWS IoT without proven identity. AWS IoT supports X.509 certificate-based authentication. Connections using MQTT use certificate-based authentication. You can attach policies to a certificate to allow or deny access to AWS IoT resources. A root CA certificate is used by your device to ensure it is communicating with AWS IoT. You can only connect your *thing* to the AWS IoT Cloud via TLS.

**Policy**

After creating a certificate for your internet-connected *thing*, you must create and attach an AWS IoT policy that will determine what AWS IoT operations the *thing* may perform. AWS IoT policies are JSON documents and they follow the same conventions as AWS Identity and Access Management policies.

You can also specify permissions for specific resources such as topics and shadows. Here is an example of the default Policy created for a new *thing*.

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [ "iot:\*" ],

"Resource": ["\*" ],

"Effect": "Allow"

}

]

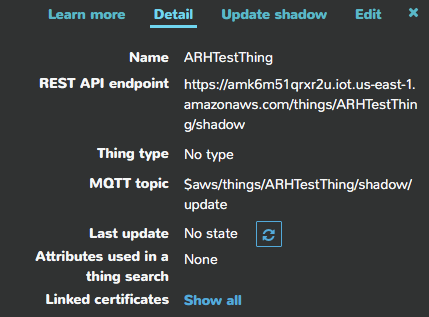
}

For the purposes of this class, when you create a *thing*, you will get a policy attached to your *thing* that will allow you to do everything you need.

### Amazon AWS MQTT

#### Internet Access

When you create an AWS IoT account, Amazon will create a new server for you in the Cloud and will turn on an MQTT Message Broker on that server. In order to connect your WICED device to that server you will need to know the IP Address of the Message Broker (to program in your firmware). You find the address by clicking on your *thing* in the console which will open up a window on the right of your browser which contains the DNS name and other information about your *thing*. (In this case amk6m51qrxr2u.iot.us-east-1.amazonaws.com.)



#### [Thing Shadow](http://docs.aws.amazon.com/iot/latest/developerguide/iot-thing-shadows.html)

A *thing* shadow (sometimes referred to as a device shadow) is a JSON document that is used to store and retrieve current state information for a *thing* (device, app, and so on). The *Thing* Shadows service maintains a *thing* shadow for each *thing* you connect to AWS IoT. You can use *thing* shadows to get and set the state of a *thing* over MQTT or HTTP, regardless of whether the *thing* is connected to the Internet. Each *thing* shadow is uniquely identified by its name.

The JSON Shadow document representing the device has the following properties:

* state:
  + desired: The desired state of the *thing*. Applications can write to this portion of the document to update the state of a *thing* without having to directly connect to a *thing*.
  + reported: The reported state of the *thing*. *Things* write to this portion of the document to report their new state. Applications read this portion of the document to determine the state of a *thing*.
* metadata: Information about the data stored in the state section of the document. This includes timestamps, in Epoch time, for each attribute in the state section, which enables you to determine when they were updated.
* timestamp: Indicates when the message was transmitted by AWS IoT. By using the timestamp in the message and the timestamps for individual attributes in the desired or reported section, a *thing* can determine how old an updated item is, even if it doesn't feature an internal clock.
* clientToken: A string unique to the device that enables you to associate responses with requests in an MQTT environment.
* version: The document version. Every time the document is updated, this version number is incremented. Used to ensure the version of the document being updated is the most recent.

An example of the document looks like this:



If you want to update the Shadow, you can publish a JSON document with just the information you want to change to the correct topic. For example, you could do:

{

"reported" : { "temp": 55 }

}

#### MQTT Topics

The AWS Message Broker will allow you to create Topics with almost any name, with one exception: Topics named “$aws/…” are reserved by AWS IoT for specific functions.

As the system designer you are responsible for defining what the topics mean and do in your system. Some [best practices](http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices) include:

1. Don’t use a leading forward slash
2. Don’t use spaces
3. Keep the topic short and concise
4. Use only ASCII characters
5. Embed a unique identifier e.g. the name of the *thing*

For example, a good topic name for a temperature sensing device might be: XXX.

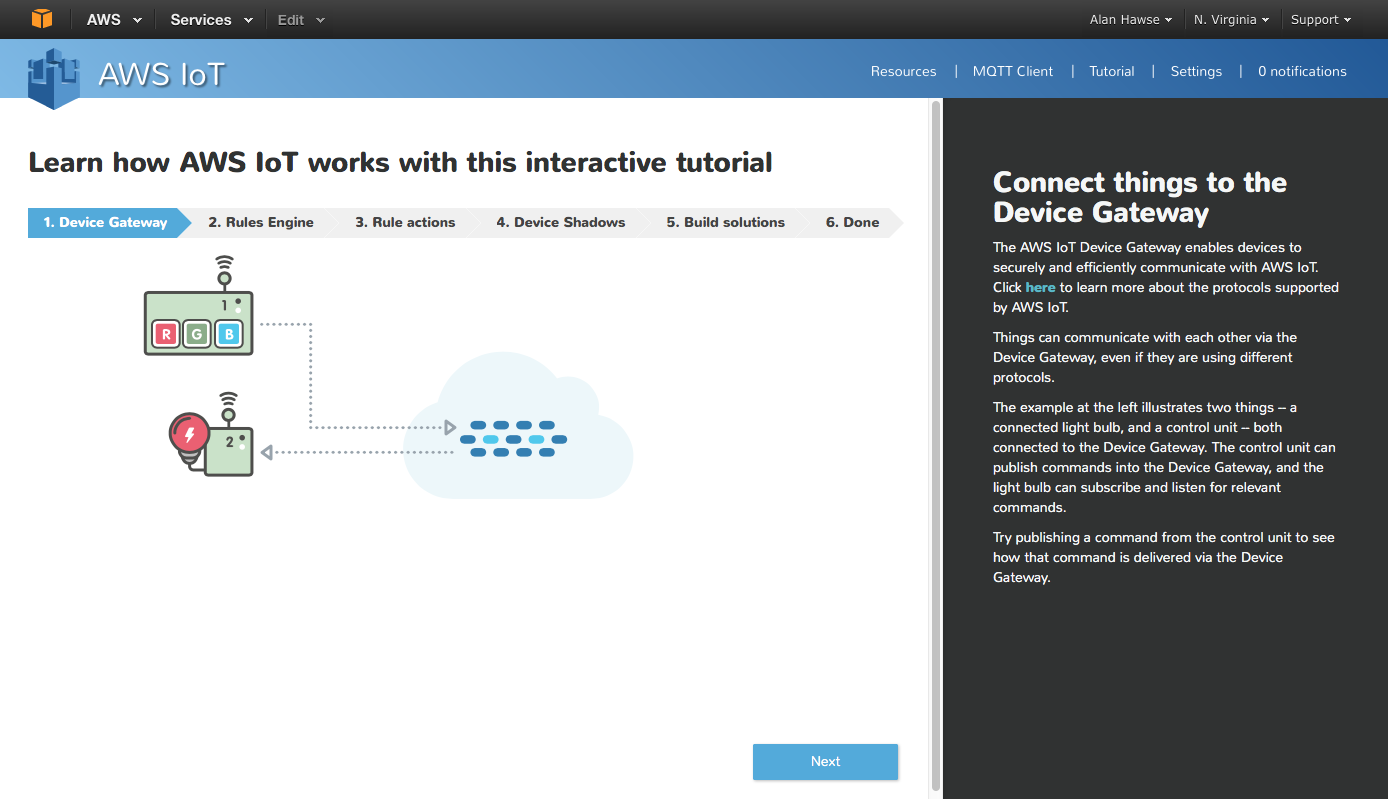
[Device Shadow MQTT Topics](https://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-mqtt.html)

Each *thing* that you have will have a group of topics of the form “$aws/things/thingName/shadow/…”

|  |  |
| --- | --- |
| **MQTT Topic Suffix** | **Function** |
| /update | The JSON message that you publish to this topic will become the new state of the shadow. |
| /update/accepted | AWS will publish a message to this topic in response to a message to /update indicating a successful update of the shadow. |
| /update/documents | When a document is updated via a publish to /update, the complete new document is published to this topic. |
| /update/rejected | AWS will publish a message to this topic in response to a message to /update indicating a rejected update of the shadow. |
| /update/delta | After a message is sent to /update, the AWS will send a JSON message with the changes in the shadow. |
| /get | If a *thing* publishes a message to this topic, AWS will respond with a message to either /get/accepted or /get/rejected with the current state of the shadow. |
| /get/accepted |
| /get/rejected |
| /delete | If a *thing* publishes a message to this topic, AWS will delete the shadow document. |
| /delete/accepted | AWS will publish to this topic when a successful /delete occurs. |
| /delete/reject | AWS will publish to this topic when a rejected /delete occurs. |

## 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 WICED apps/demo/aws\_iot/pub\_sub/publisher project to your own directory (i.e. wa101/07b/02\_publisher and update all of the files.
2. Modify the DCT for your network
3. Copy the certificates that you generated in (01) 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. Open the serial port and watch your terminal session
8. Subscribe to the topic using the AWS MQTT client (and test publishing). When you press the button you should see updates to the topic.

### 03 Explain in detail the firmware flow for the 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 messages 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, 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 Build and test the demo.aws\_iot.pub\_sub.subscriber App

1. Copy the WICED application from apps/demo/aws\_iot/pub\_sub/subscriber to your directory (i.e. wa101/07b/05\_subscriber) and modify 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* (you can share credentials).

### 07 Build and test the Shadow App

1. Copy the WICED application from apps/demo/aws\_iot/shadow to your directory and update the Makefile
2. Update the DCT to have a SoftAP name that is unique (so as not to collide with others in your class)
3. Update the message broker address and the MQTT Topic
4. Attach to the Soft AP on your board from your computer’s WiFi
   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 instructions 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?”

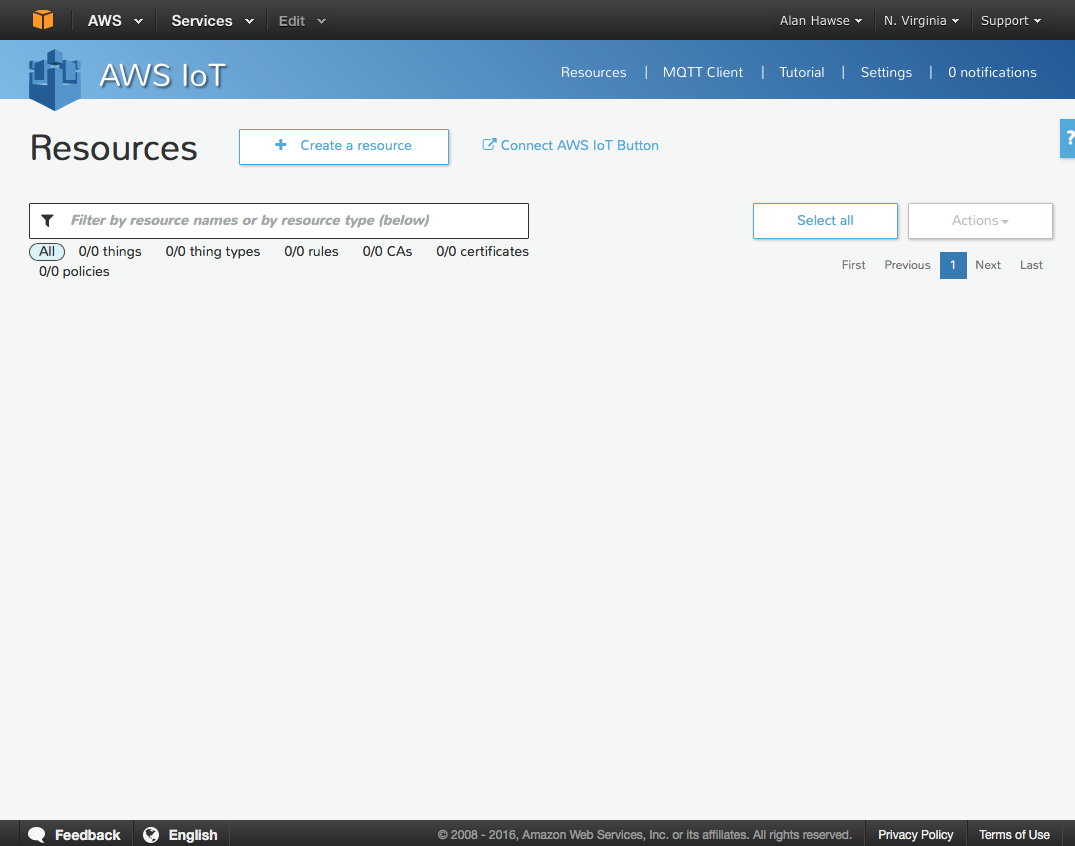
## References

|  |  |
| --- | --- |
| **Resources** | **Link** |
| [AWS Developers Guide](http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html) | http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html |
| [AWS IOT Getting Started](https://aws.amazon.com/iot/getting-started/) | https://aws.amazon.com/iot/getting-started/ |
| [A nice powerpoint about MQTT](http://www.slideshare.net/PeterREgli/mq-telemetry-transport) | http://www.slideshare.net/PeterREgli/mq-telemetry-transport |
| [MQTT Topic Naming Best Practices](http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices) | http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices |
| [Avnet Getting Started](http://cloudconnectkits.org/system/files/GSG-BCM4343W%20IoT%20Starter%20Kit%20-%20Getting%20Started%20%28v1.1%29.pdf) | http://cloudconnectkits.org/system/files/GSG-BCM4343W%20IoT%20Starter%20Kit%20-%20Getting%20Started%20%28v1.1%29.pdf |
| Avnet User Guide [Part1](http://cloudconnectkits.org/system/files/Tutorial%20Part1%20-%20Tool%20Install%2C%20USB%20drivers%20and%20AWS%20Shadow%20%28v1.1%29.pdf) and [Part2](http://cloudconnectkits.org/system/files/Tutorial%20Part2-App%20Development%20using%20WICED%20SDK-v1.1.pdf) |  |
| [AWS Forum](https://forums.aws.amazon.com/forum.jspa?forumID=210) | https://forums.aws.amazon.com/forum.jspa?forumID=210 |
|  |  |
|  |  |
|  |  |
|  |  |

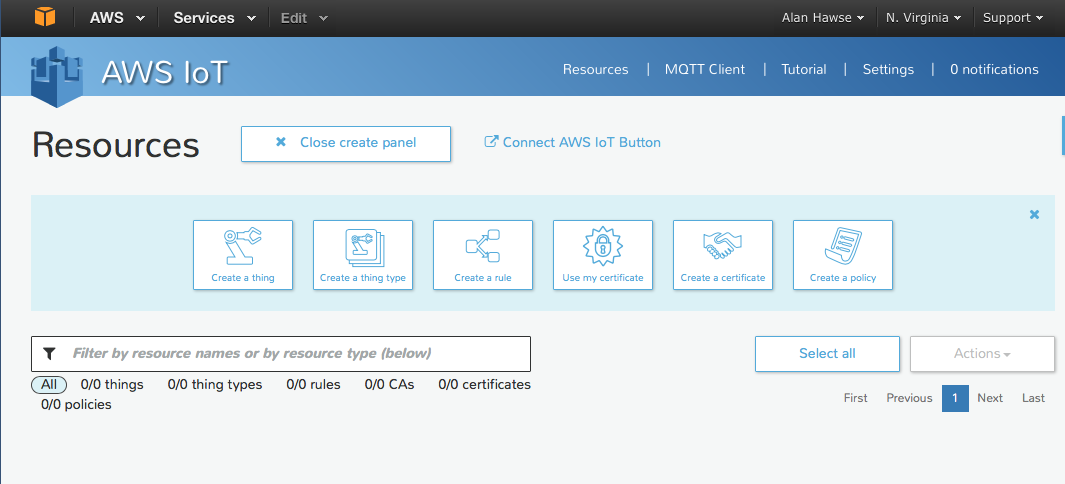
## Related Example “Apps”

## Procedure to Provision an AWS IoT *Thing*

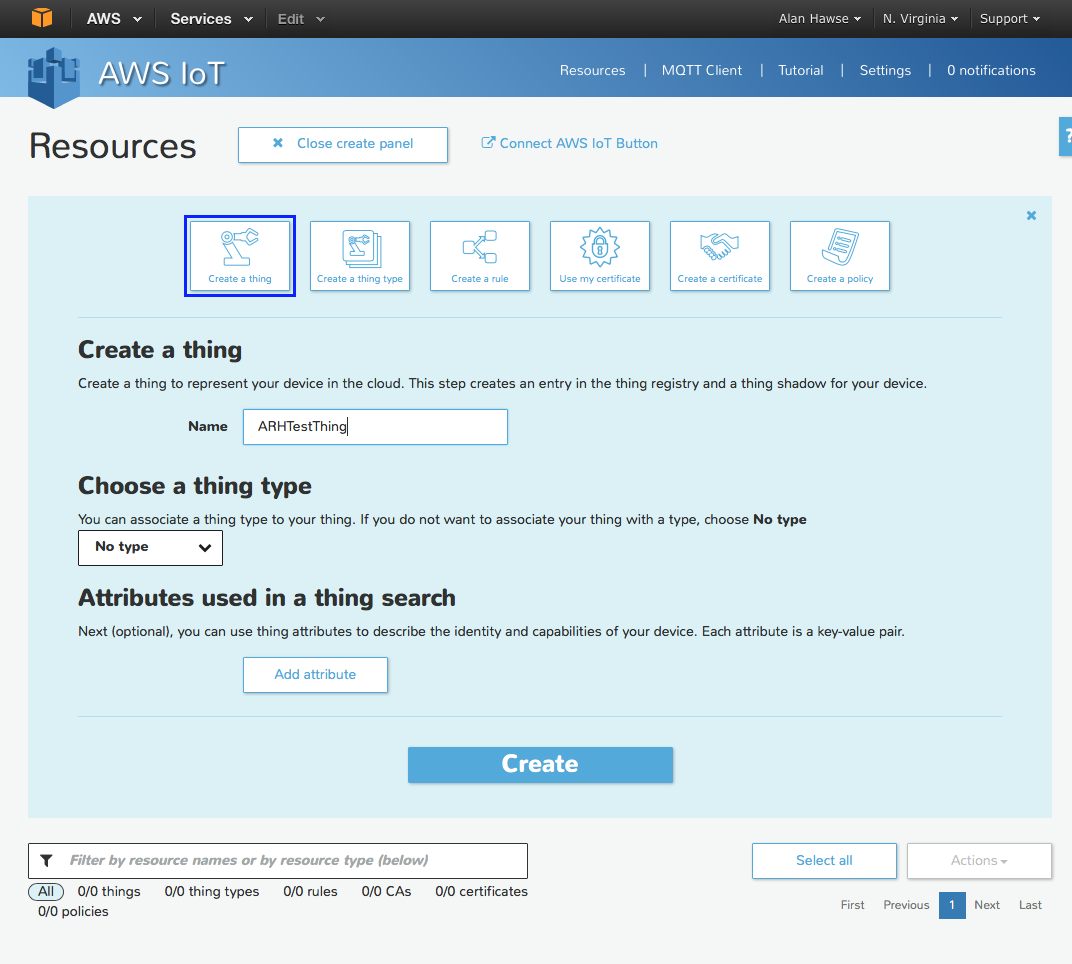
1. Start by going to the AWS IoT Console which will start blank. Press “Create a resource”.



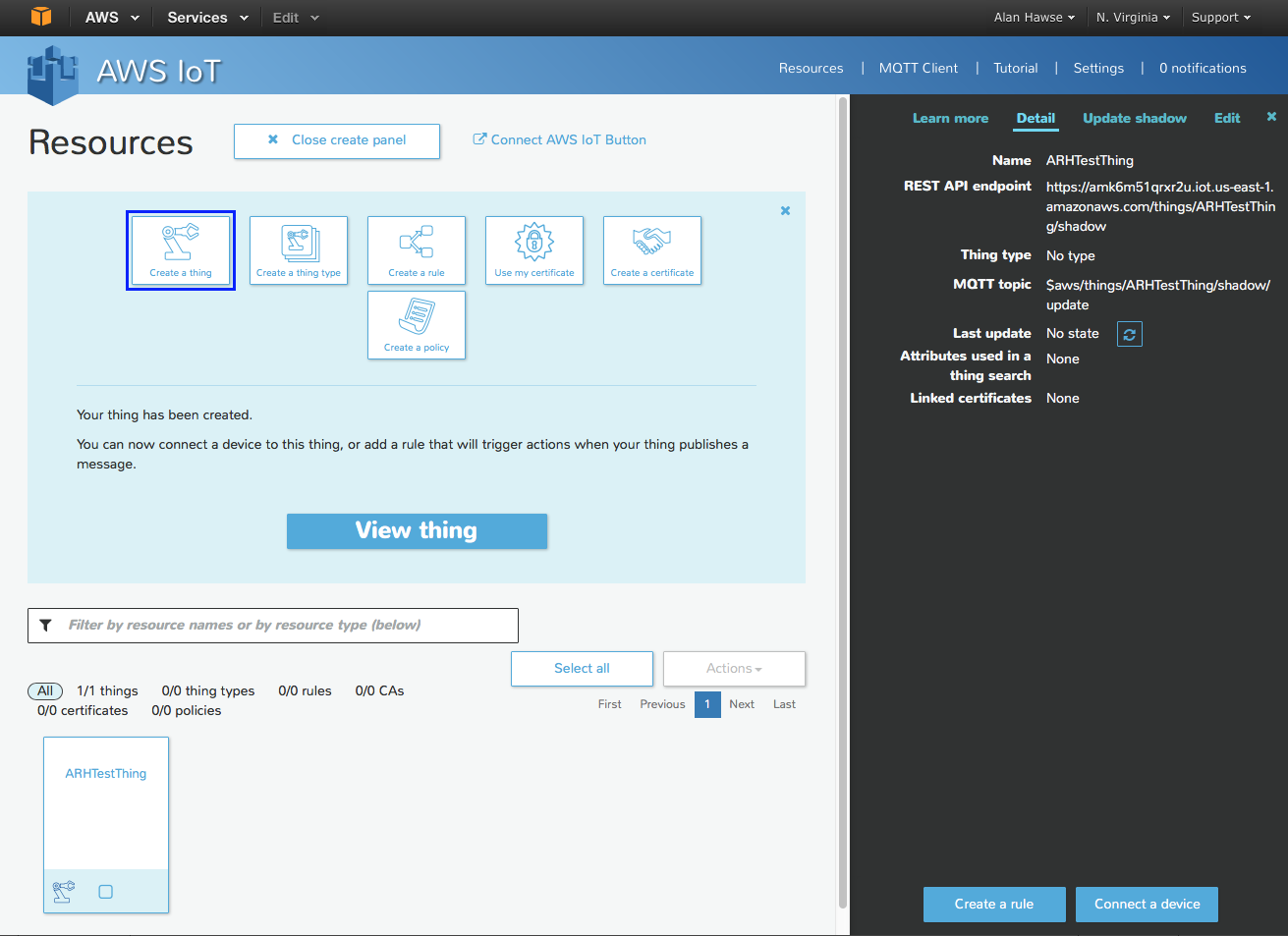
1. Press “Create a *thing*”.



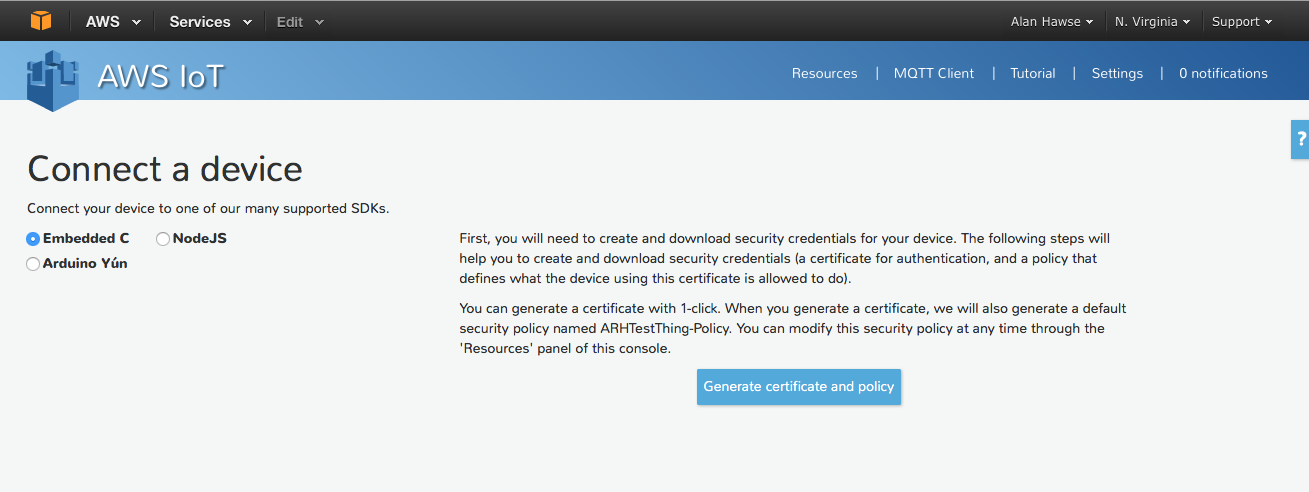
1. Name it “ARHTestThing” (or whatever) and press “Create”.



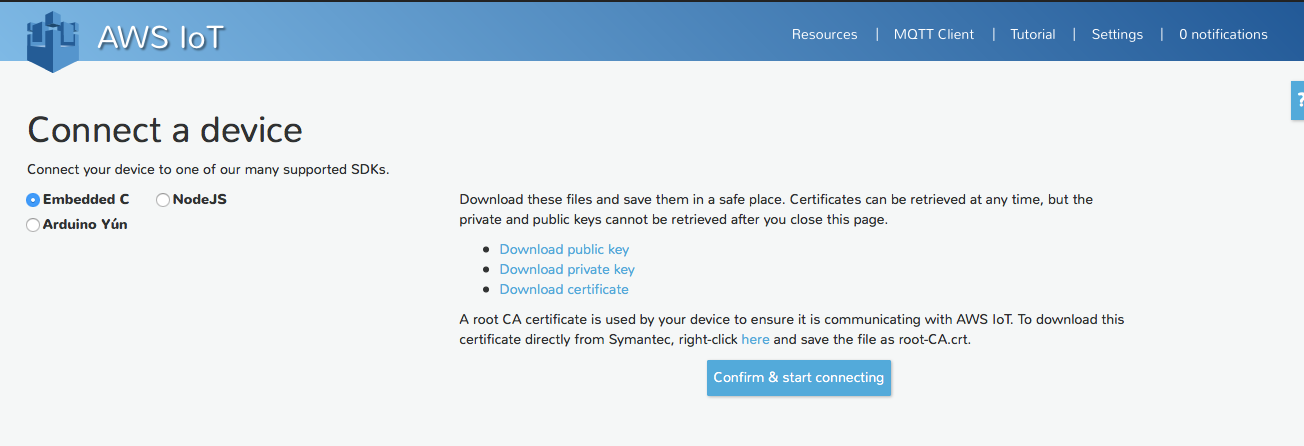
1. Click on your *thing* and it will open a window up to the right. Before you can do anything with the *thing* you need to create the encryption keys that enable you to identify yourself as that *thing*, and then update the data. To do this, click on “Connect Device” (in the lower right hand corner).



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

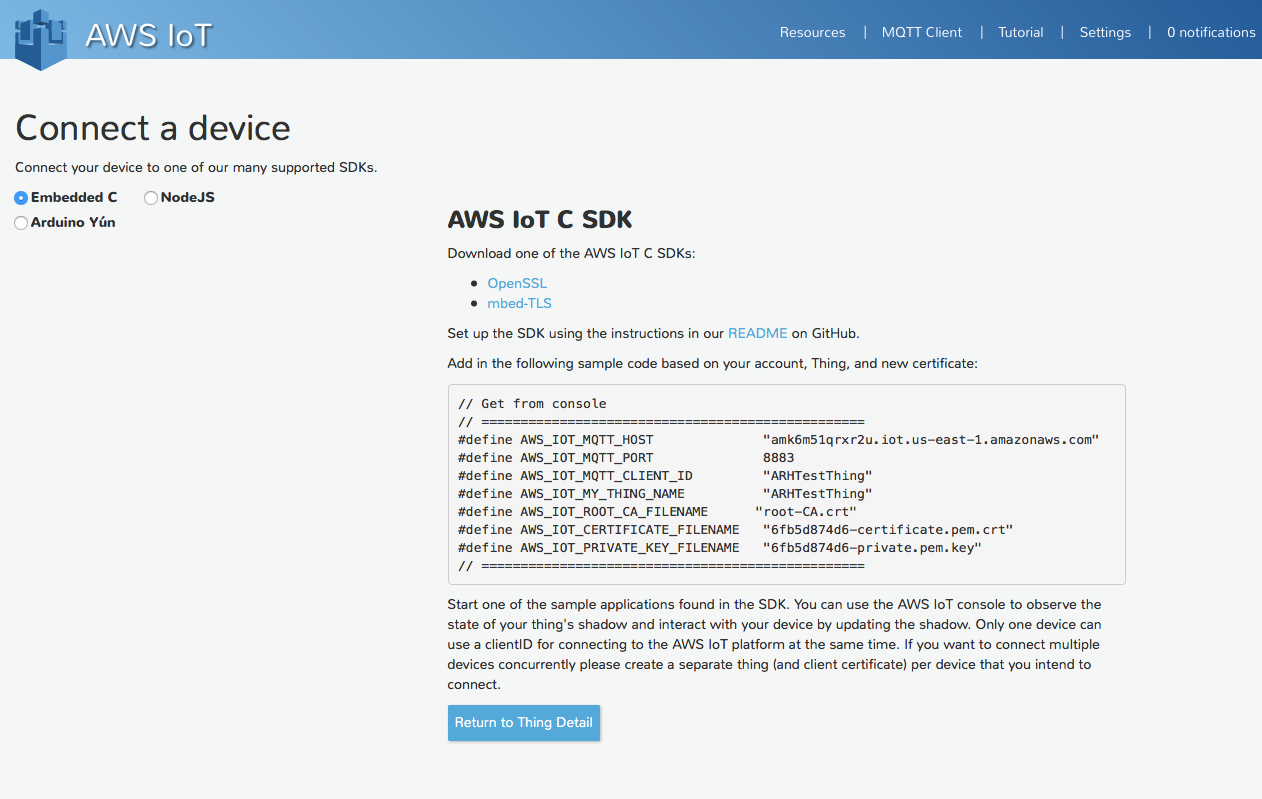


1. Now you need to download the “public key”, “private key” and “certificate”. If you forget this step you cannot come back… so really you must download those files now to make the TLS work! Once you have downloaded the keys, then click on “Confirm and start connecting”.



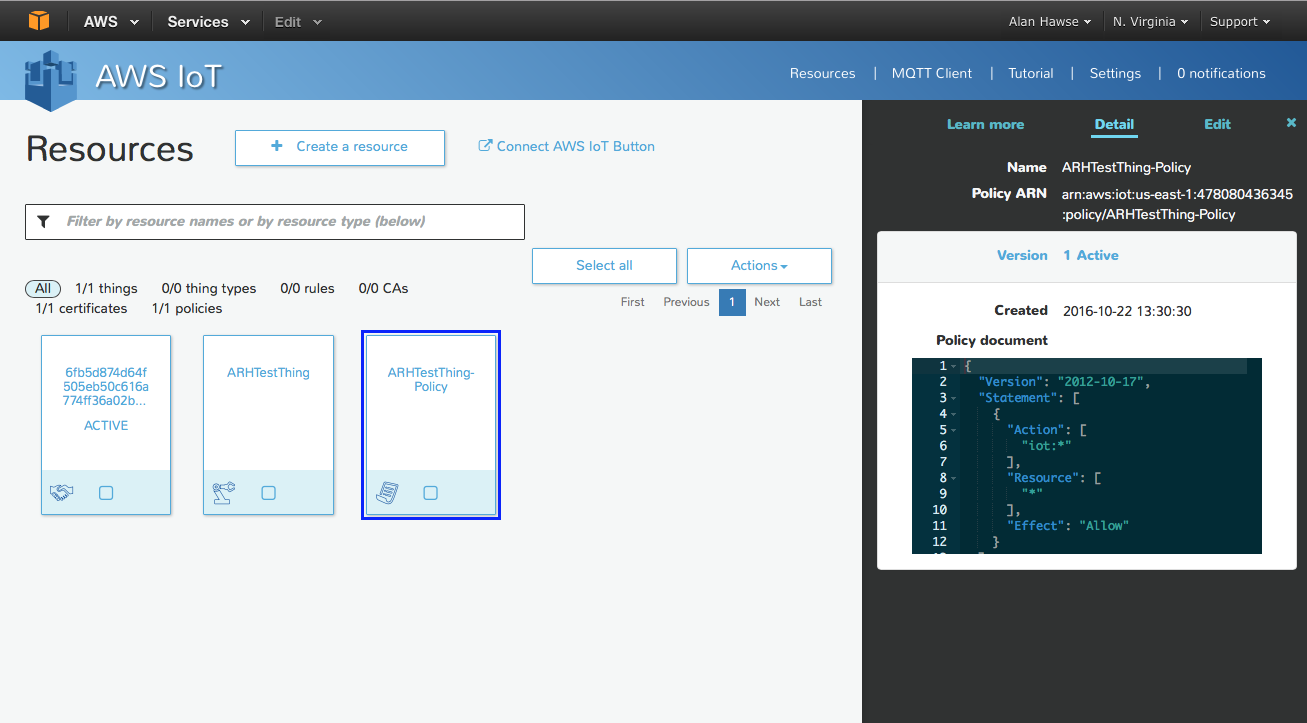
**Do this first!**

1. Save the Embedded C (copy/paste) if you want to use it then press “Return to Thing Detail”.



1. Click on the Certificate which will show you that the certificate, policy and *thing* are all linked.

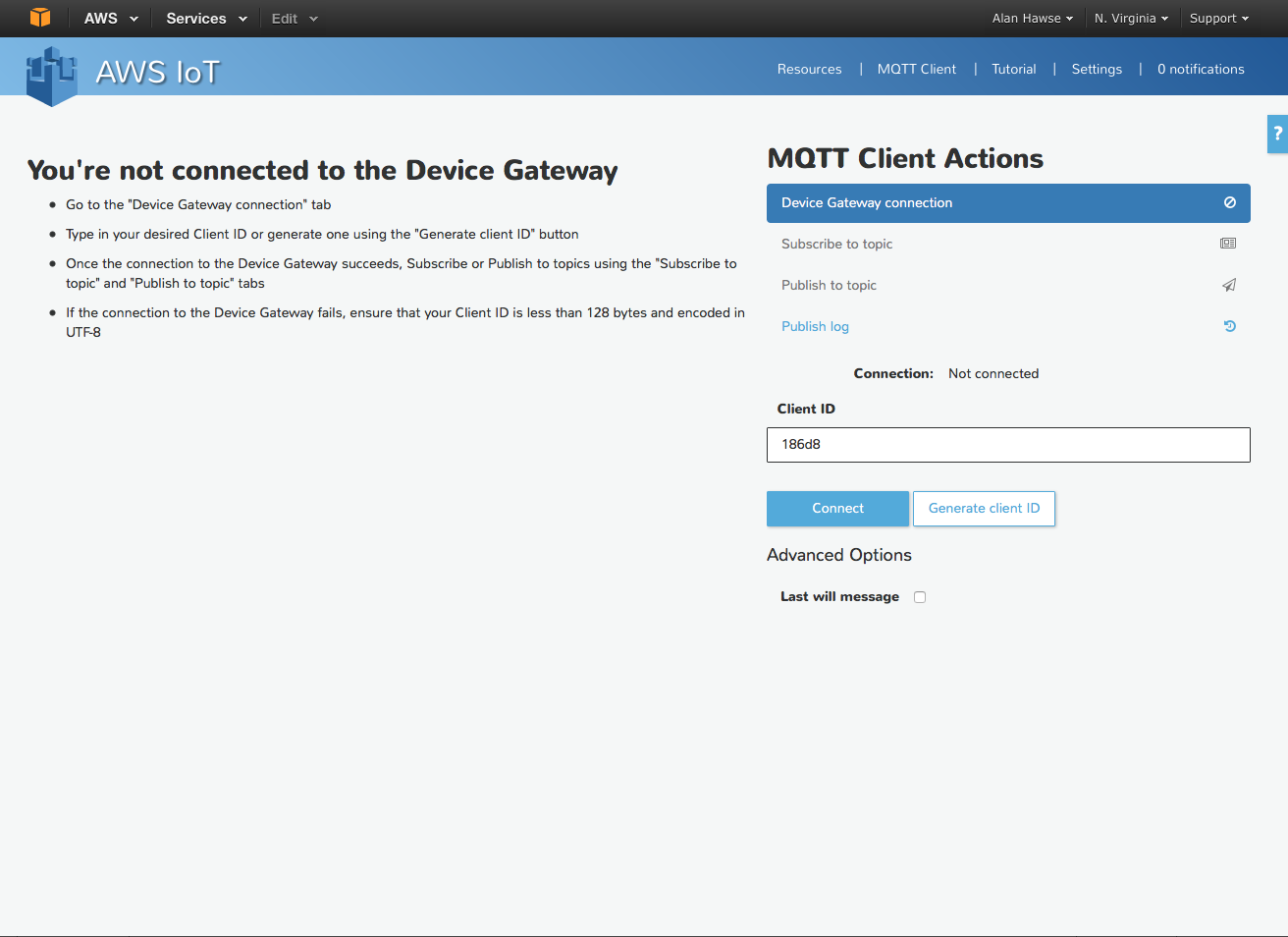


1. Click on your *thing* policy. You can see the Actions you are allowed to take on which resources. In this case the *thing* is allowed to do an “iot:” action. 

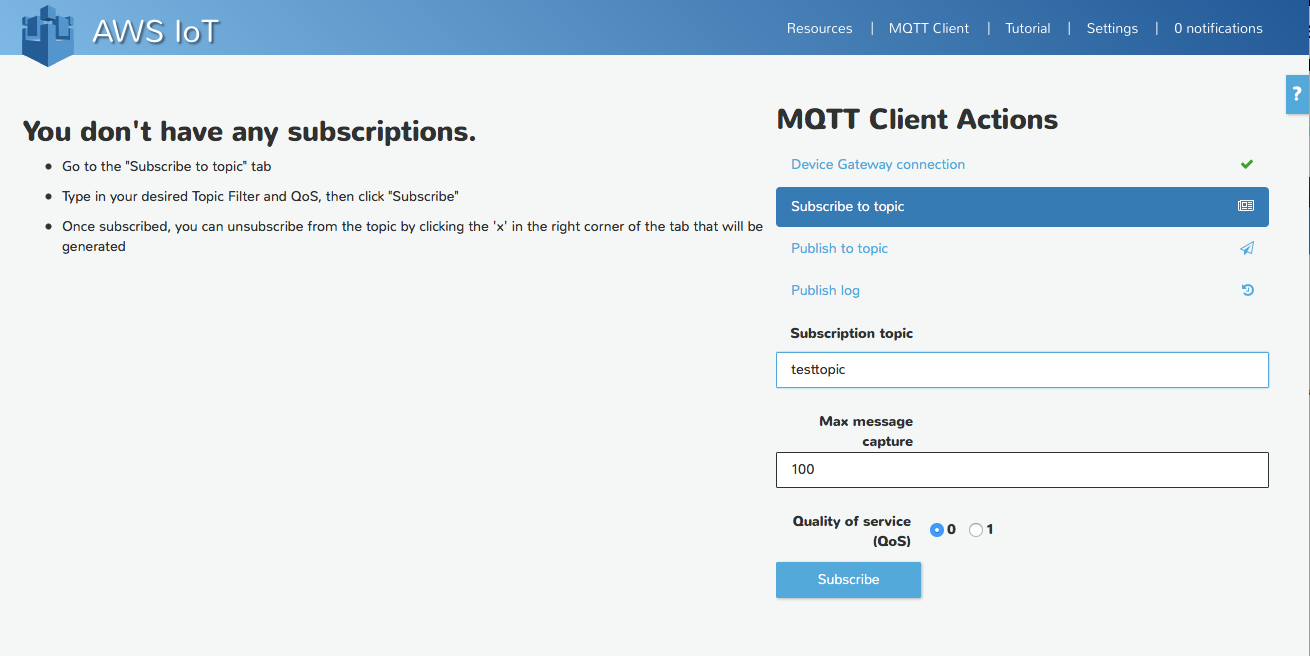
## Procedure to use the AWS MQTT Client

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:

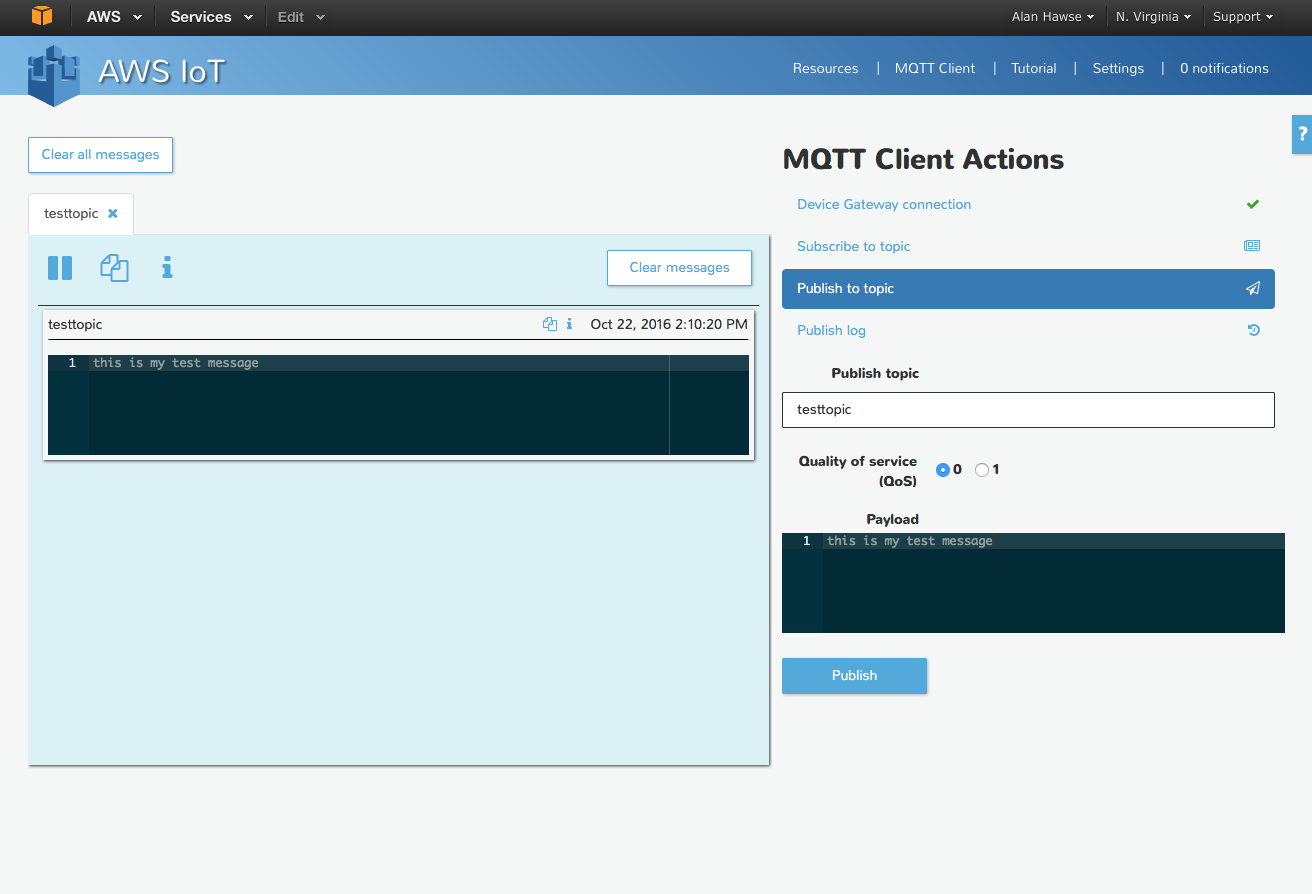
1. Click on “MQTT Client”. Then give your client a unique ID (like ARHTestThing from the previous example but don’t use that name as it is already assigned) or you can click “Generate client ID” and AWS will think up a random-ish ID (which is what I did in the screen shot below). Once you have a name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



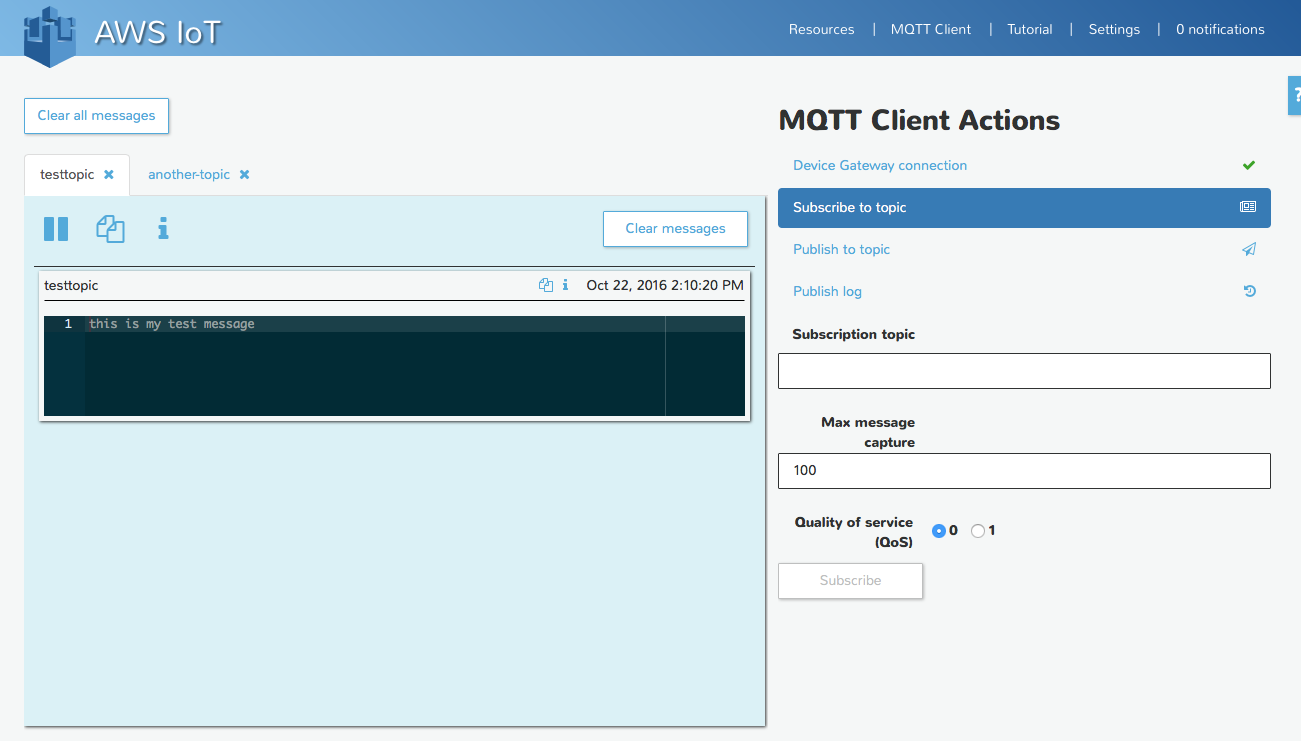
1. Now that you have a connection, the first thing to do is Subscribe to a topic. Remember that there is nothing particularly special about the topic name. Let’s choose the topic “testtopic”



1. Now that I am subscribed to a topic I can publish 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” (Exercise 02)

