

Chapter 7B: Amazon Web Services (AWS)

At this end of Chapter 7B you will understand:

How the [AWS Cloud works](#) including:

- How to provision “things” (which for semantic reasons, will be notated *thing*, a.k.a. your IoT device, in this chapter) in the AWS IoT Cloud by creating a *thing*, policies and certificates.
- AWS Security
- The *thing* shadow
- How to use the AWS IoT test Client to subscribe and publish to topics
- Understand the scope of systems that can be implemented in the AWS Cloud (SNS, Database etc.)

7B.1	AMAZON WEB SERVICES (AWS)	2
7B.2	AWS IOT INTRODUCTION	3
7B.3	AWS IOT RESOURCES	4
7B.4	AWS IOT CONSOLE	5
	7B.4.1 CREATING AN AWS IOT ACCOUNT.....	5
	7B.4.2 THING SHADOW	6
	7B.4.3 TOPICS	7
	7B.4.4 DEVICE SHADOW TOPICS	8
7B.5	EXERCISE(S)	9
	EXERCISE - 7B.1 RUN AWS TUTORIAL	9
	EXERCISE - 7B.2 CREATE NEW AWS THING	11
	EXERCISE - 7B.3 LEARN HOW TO USE THE AWS MQTT TEST CLIENT.....	16
7B.6	REFERENCES	18

7B.1 Amazon Web Services (AWS)

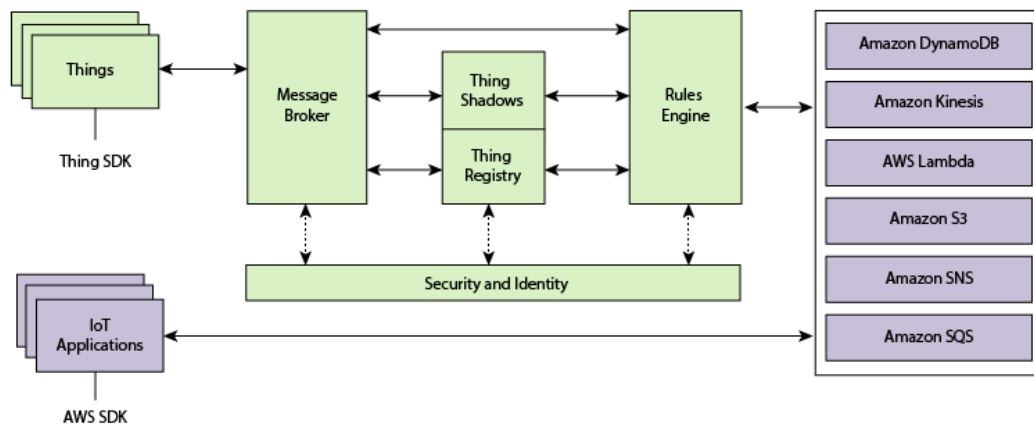
[AWS](#) is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality (which makes more money for Amazon than their retail operations). AWS is built from a vast array of both virtual and actual servers and networks as well as a boatload of webserver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

- [AWS IoT](#): A cloud platform that provides Cloud services for IoT devices (the subject of this chapter).
- Amazon Elastic Cloud ([EC2](#)/[EC3](#)): A virtualized compute capability, basically Linux, Windows etc. servers that you can rent.
- [Amazon Lambda](#): A Cloud service that enables you to send event driven tasks to be executed.
- Storage: Large fast file systems called [Amazon S3](#) & [AWS Elastic File System](#).
- Databases: Large fast databases called [Dynamo DB](#), [Amazon Relational Database \(RDS\)](#), [Amazon Aurora](#).
- 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 \(SNS\)](#): A platform to send messages including SMS and Email.
- [Amazon Simple Queueing Services \(SQS\)](#): A platform to send messages between servers (NOT the same thing a MQTT messages).
- [Amazon Kinesis](#): A platform to stream and analyze “massive” amounts of data. This is the plumbing for AWS IoT.

7B.2 AWS IoT Introduction

The AWS IoT Cloud service supports MQTT Message Brokers, HTTP access, **plus** a bunch of server side functionality that provides:

- Message Broker: A virtual MQTT Message Broker.
- A virtual HTTP Server.
- 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.



7B.3 AWS IoT Resources

There are three types of resources in AWS: *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 a proven identity. AWS IoT supports X.509 certificate-based authentication. Connections to AWS use certificate-based authentication. You should attach policies to a certificate to allow or deny access to AWS IoT resources. A root CA (certification authority) certificate is used by your device to ensure it is communicating with the actual Amazon Web Services site. 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 specify permissions for specific resources such as topics and shadows. Here is an example of a Policy created for a new *thing* that allows any IoT action for any resource.

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

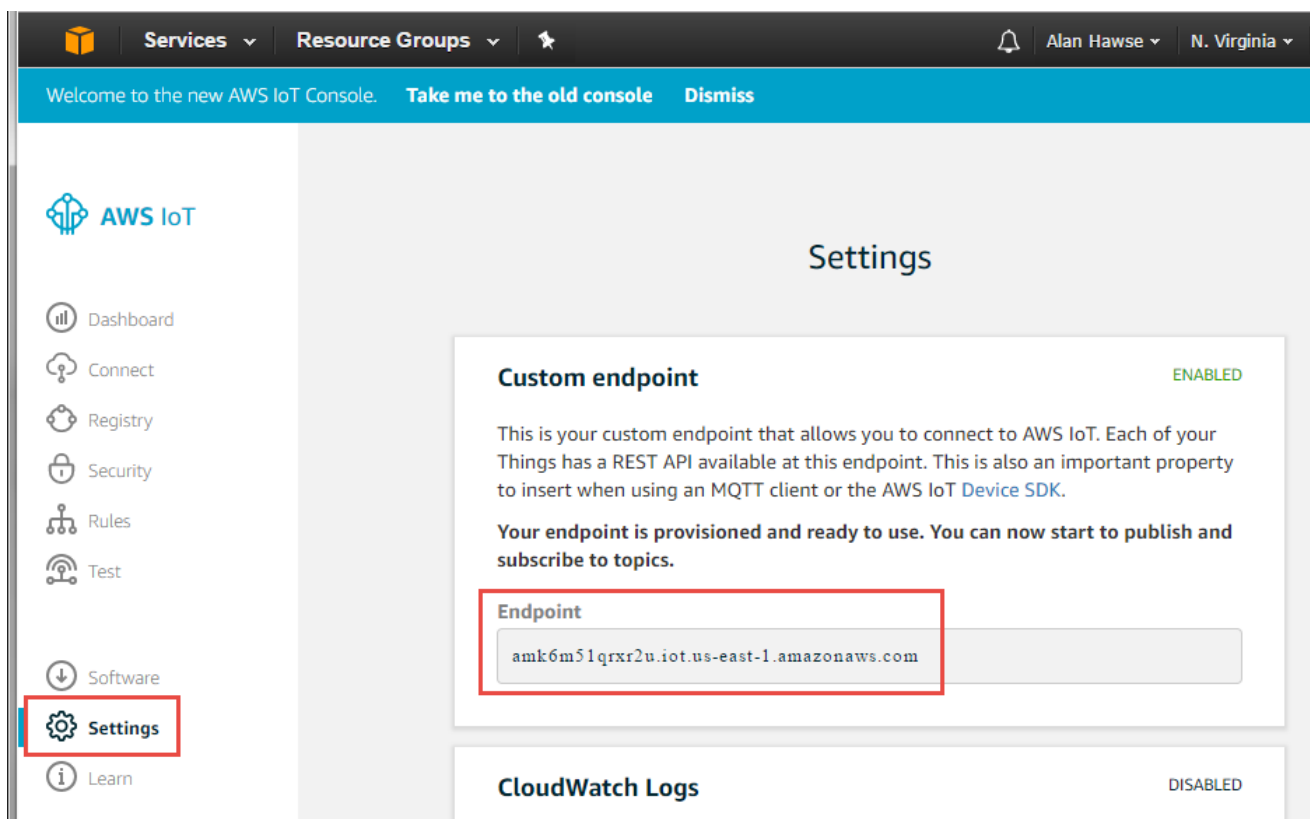
7B.4 AWS IoT Console

7B.4.1 Creating an AWS IoT Account

To create a new AWS account, you need to provide a credit card number. The basic account is free for a year but if you don't cancel before that (or remove your credit card from the Amazon payment options) it will start charging your credit card after a year. For that reason, we have setup a class AWS account that you can use for the exercises. However, the password for that account will be changed after the class is over and any *things* you create there will be deleted. If you want to continue to use AWS after the class you will need to setup your own account.

When you create an AWS IoT account, Amazon will create a new virtual machine for you in the Cloud and will turn on an MQTT Message Broker and HTTP server on that machine. To connect your WICED device to the machine you will need to know the DNS name of the virtual machine.

To find the virtual machine's DNS name, click on "Settings" at the lower left corner of the main console window. The name is listed as the Endpoint.



7B.4.2 [Thing Shadow](#)

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 currently 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 a shadow document looks like this:

```
{
  "state": {
    "desired": {
      "color": "RED",
      "sequence": [ "RED", "GREEN", "BLUE" ]
    },
    "reported": {
      "color": "GREEN"
    }
  },
  "metadata": {
    "desired": {
      "color": {
        "timestamp": 12345
      },
      "sequence": {
        "timestamp": 12345
      }
    },
    "reported": {
      "color": {
        "timestamp": 12345
      }
    }
  },
  "version": 10,
  "clientToken": "UniqueClientToken",
  "timestamp": 123456789
}
```

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:

```
{
  "state": {
    "reported": {
      "color": "BLUE"
    }
  }
}
```

Note that spaces and carriage returns are optional, so the above could be written as:

```
{"state":{"reported":{"color": "BLUE"}}
```

7B.4.3 Topics

You can interact with AWS using either MQTT or HTTP. While topics are an MQTT concept, you will see later that topic names are important even when using HTTP to interact with *thing* shadows. 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](#) 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: myDevice/temperature.

7B.4.4 Device Shadow Topics

Each *thing* that you have will have a group of topics of the form “\$aws/things/<thingName>/shadow/<type>” which allow you to publish and subscribe to topics relating to the shadow. The specific shadow topics that exist are:

MQTT Topic Suffix <type>	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 if the desired state and the reported state are not equal. The message contains all attributes that don't match.
/get	If a <i>thing</i> 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 <i>thing</i> 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.

The update topic is useful when you want to update the state of a *thing* on the cloud. For example, if you have a *thing* called “myThing” and want to update a value called “temperature” to 25 degrees in the state of the thing, you would publish (for MQTT) or POST (for HTTP) using the following topic and message:

topic: \$aws/things/myThing/shadow/update

message: {"state":{"reported":{"temperature":25}}}

Once the message is received, the MQTT message broker will publish to the /accepted, and /documents topics with the appropriate information.

If you are using the MQTT test server to subscribe to topics, you can use “#” as a wildcard at the end of a topic to subscribe to multiple topics. For example, you can use “\$aws/things/theThing/shadow/#” to subscribe to all shadow topics for the *thing* called “theThing”.

You can also use “+” as a wildcard in the middle of a topic to subscribe to multiple topics. For example, you can use “\$aws/things+/shadow/update” to subscribe to update topics for all *thing* shadows.

7B.5 Exercise(s)

Exercise - 7B.1 Run the AWS Tutorial

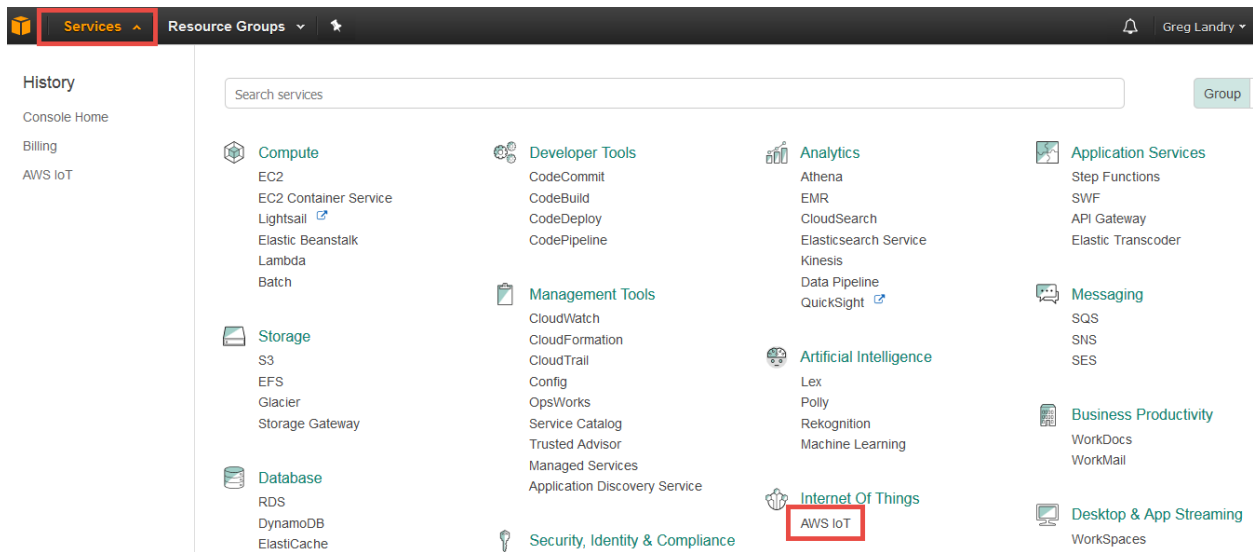
Run the tutorial on the Amazon IoT Console (console.aws.amazon.com)

Sign up for an AWS account or use the class server. The login for the class server is:

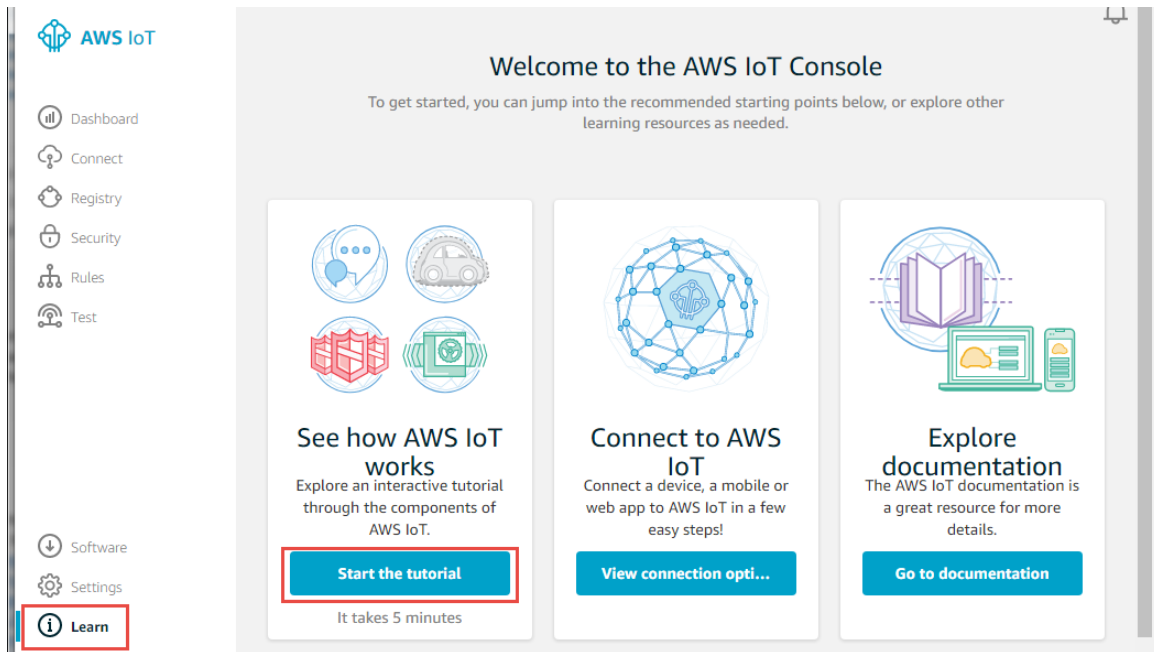
ID: arh@cypress.com

Password: See the back cover of manual for the current password.

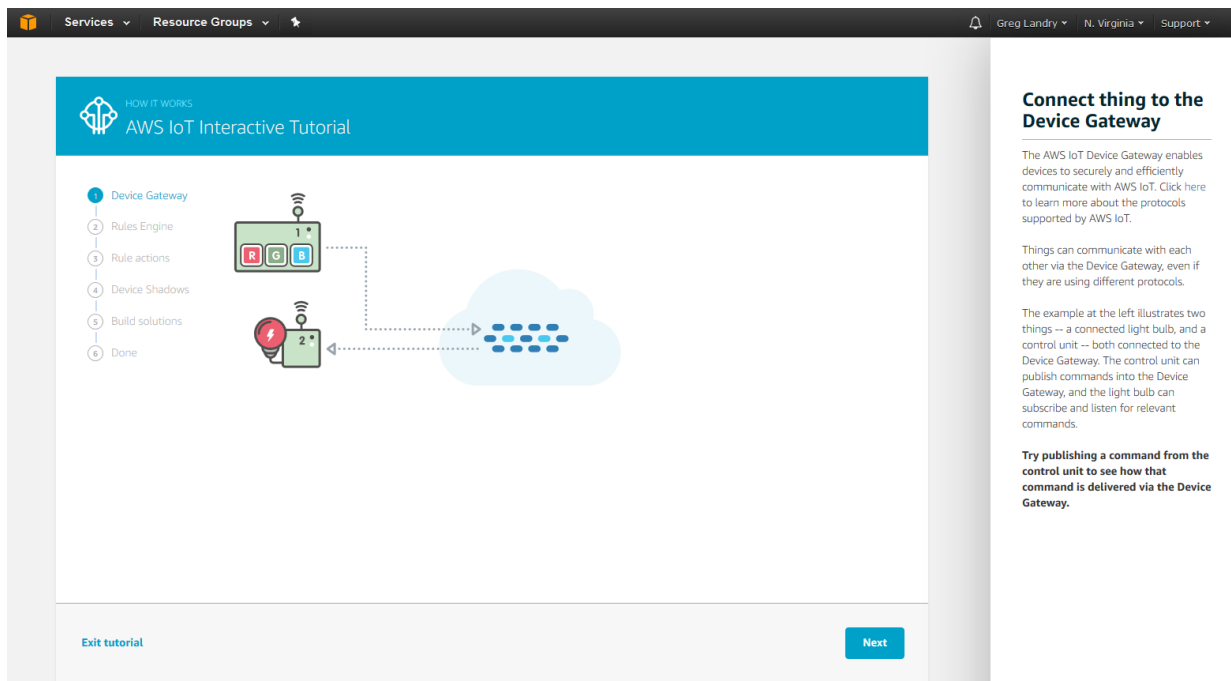
From the Services menu, select “AWS IoT”:



In the lower-left corner of the IoT screen click on “Learn” and then click “Start the tutorial”:



Follow the instructions to complete the tutorial.

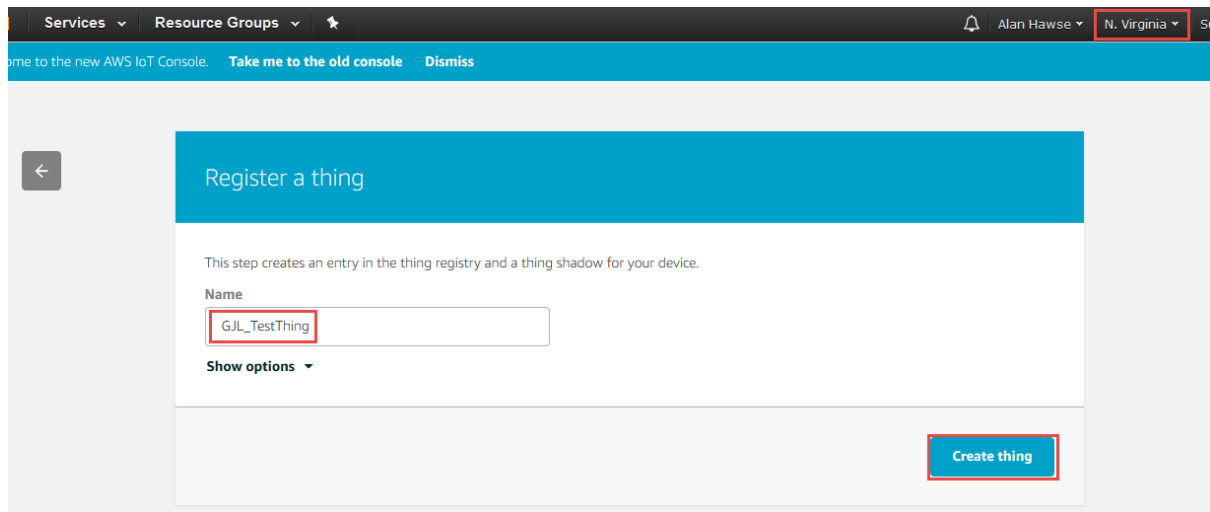


Exercise - 7B.2 Create new AWS Thing

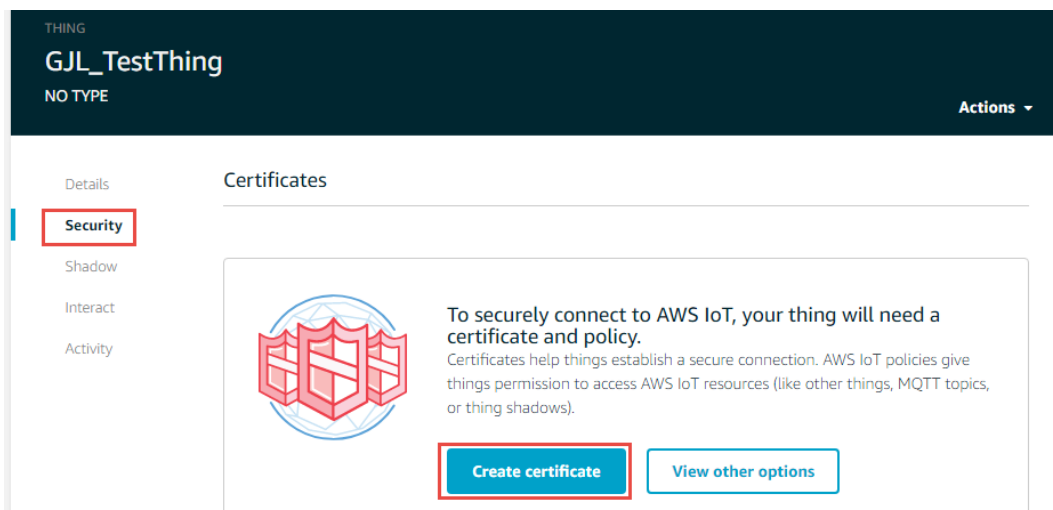
Provision a new *thing* in the AWS IoT Cloud, and establish its policy and credentials.

Note: The steps below assume that you are using the existing class AWS account. If you create your own account the steps may be slightly different but will still follow the same flow.

- Once you have watched the tutorial, you should be on the “Register a thing” page.
Hint: The example projects use US East time zone 1 (N. Virginia). If you create your own AWS account and use a different time zone, you will need to search for “us-east-1” in the source code for each project in the later exercises and update as necessary.
- Name your *thing* “<YourInitials>_TestThing” (or whatever) and press “Create thing”.

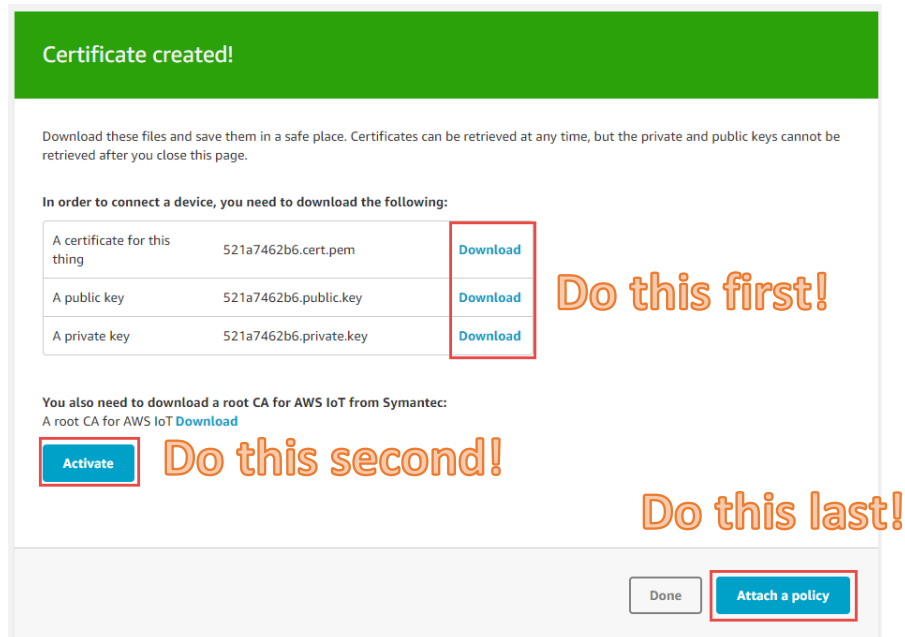


- Before you can access the broker from your WICED kit you need to create the encryption keys that enable you to identify it as an allowed device. To do this, from the *thing* page click on Security and then on Create Certificate.



4. Now you need to download the “certificate”, “public key” and “private key”. If you forget this step you cannot come back...so really you must download those files now to make the TLS work! You must also “Activate” the certificate.
5. You should also write down the certificate ID since you may need it later when you attach a policy to the certificate.

Note: The window also has an option to download a root CA for AWS IoT from Symantec (a trusted certification authority). However, you don’t need to do this since the root CA for AWS IoT is already included in the WICED SDK.



Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

A certificate for this thing	521a7462b6.cert.pem	Download
A public key	521a7462b6.public.key	Download
A private key	521a7462b6.private.key	Download

Do this first!

You also need to download a root CA for AWS IoT from Symantec:
A root CA for AWS IoT [Download](#)

[Activate](#) **Do this second!**

Do this last!

[Done](#) [Attach a policy](#)

6. Click “Attach a policy” and then click on “Create new policy”.

7. Give the new policy a name such as "<YourInitials>_TestThing_Policy". Add the action as "iot:*", use "*" for the Resource ARN, and select "Allow". Then click the "Create" button.

Create a policy

Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters).

Name

GJL_TestThing_Policy

Add statements

Policy statements define the types of actions that can be performed by a resource. **Advanced mode**

Action

iot:*

Resource ARN

*

Effect

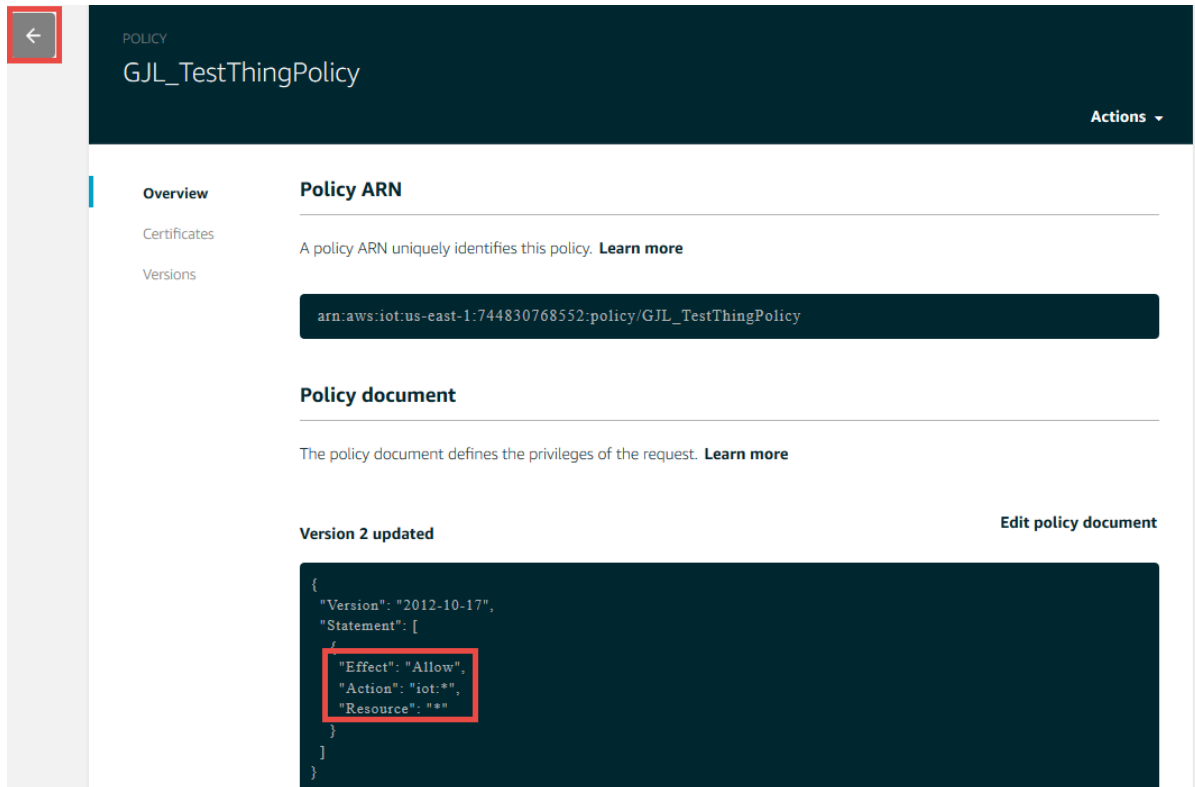
☒ Allow ☐ Deny

Remove

Add statement

Create

8. You will now see the policy document details. In this case, any IoT operation (iot:*) is allowed for any resource (*).



POLICY
GJL_TestThingPolicy

Actions ▾

Overview

Certificates

Versions

Policy ARN

A policy ARN uniquely identifies this policy. [Learn more](#)

arn:aws:iot:us-east-1:744830768552:policy/GJL_TestThingPolicy

Policy document

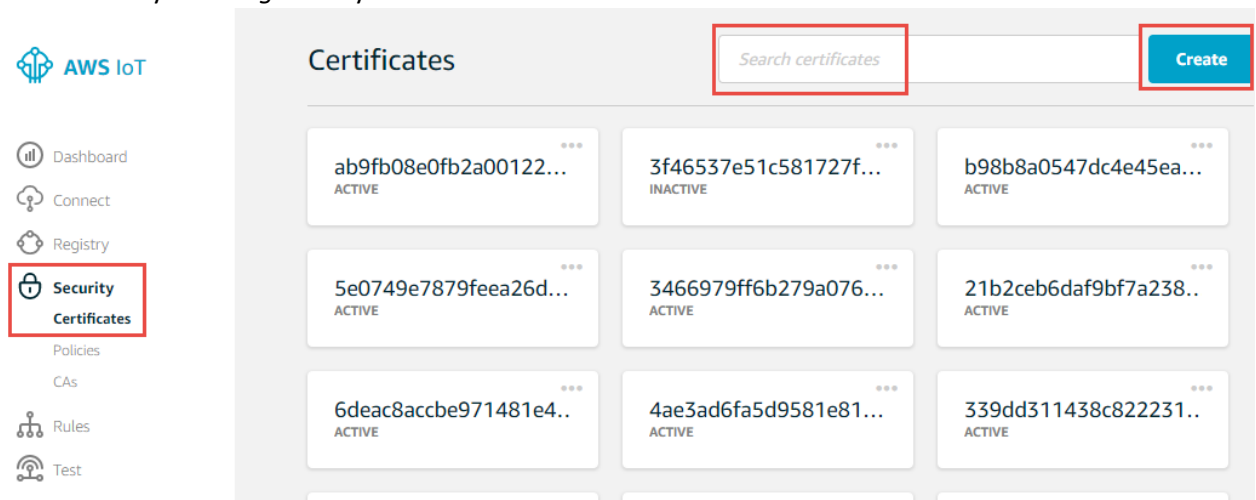
The policy document defines the privileges of the request. [Learn more](#)

Version 2 updated [Edit policy document](#)

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

9. You now need to attach the policy to the certificate. First click the left arrow on the left side of the screen show above. Then select Security -> Certificates from the left panel, and click on your certificate.

Note that you can use the search box in the upper right corner to find your certificate by name. In fact, you can even enter your *thing* name in the box and it will find the certificate that was attached to your *thing* when you first created it.



AWS IoT

Dashboard

Connect

Registry

Security

Certificates

Policies

CAs

Rules

Test

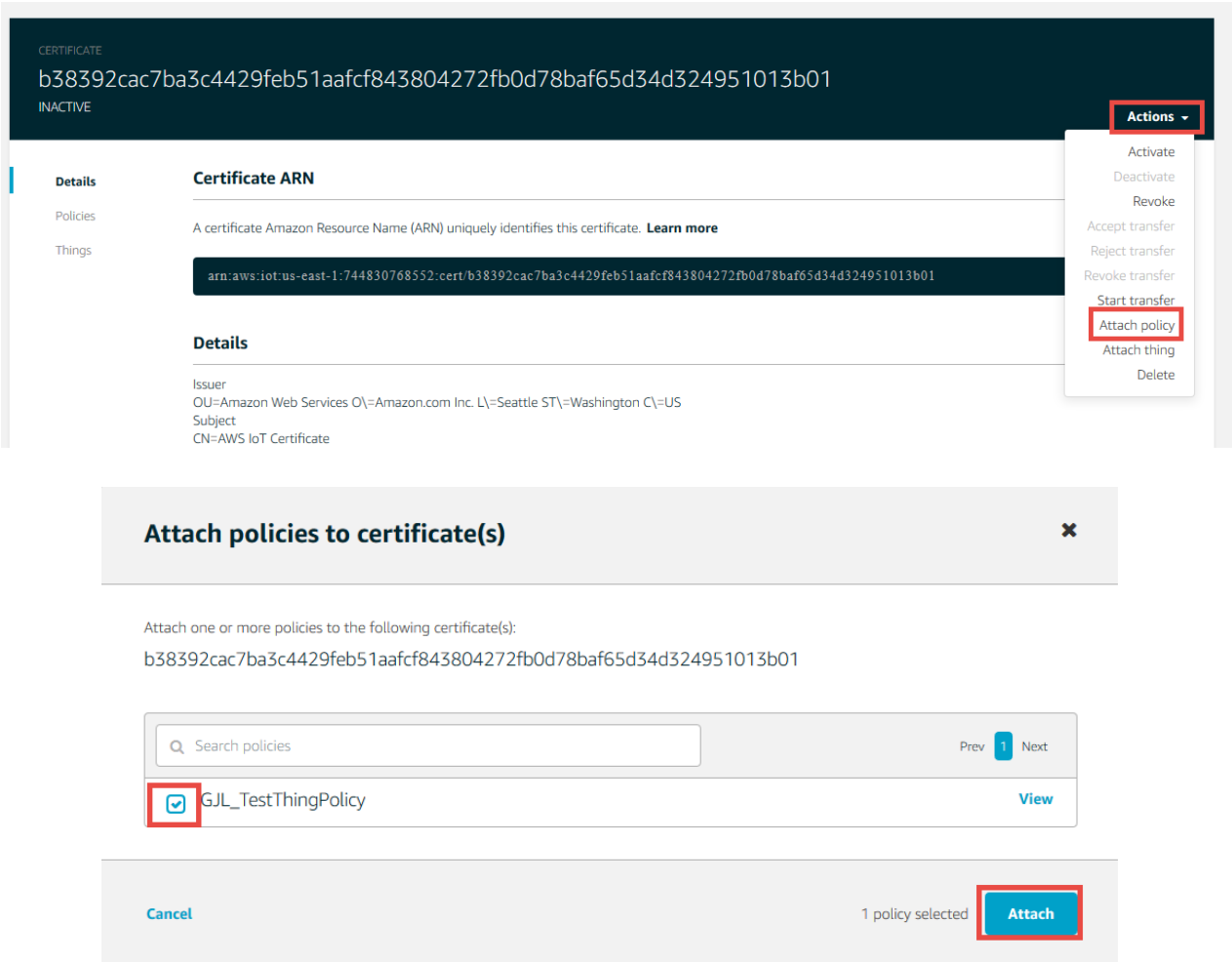
Certificates

Search certificates

Create

ab9fb08e0fb2a00122... ACTIVE	3f46537e51c581727f... INACTIVE	b98b8a0547dc4e45ea... ACTIVE
5e0749e7879feea26d... ACTIVE	3466979ff6b279a076... ACTIVE	21b2ceb6daf9bf7a238.. ACTIVE
6deac8accbe971481e4.. ACTIVE	4ae3ad6fa5d9581e81... ACTIVE	339dd311438c822231.. ACTIVE

10. Once you click on your certificate, select “Actions -> Attach Policy”. Select your policy and click “Attach”. Click on the left arrow in the upper left when you are done to return to the AWS IoT main page.



The screenshot shows the AWS IoT console interface. At the top, a dark blue header displays the certificate ID `b38392cac7ba3c4429feb51aafc843804272fb0d78baf65d34d324951013b01` and its status as **INACTIVE**. A sidebar on the left contains links for **Details**, **Policies**, and **Things**. The main content area shows the **Certificate ARN** and its details, including the issuer (Amazon Web Services) and subject (AWS IoT Certificate). An **Actions** dropdown menu is open on the right, with options like **Activate**, **Deactivate**, **Revoke**, **Accept transfer**, **Reject transfer**, **Revoke transfer**, **Start transfer**, **Attach policy** (highlighted), **Attach thing**, and **Delete**. Below the screenshot, a modal window titled **Attach policies to certificate(s)** is shown. It contains a search bar for policies, a list of policies with **GJL_TestThingPolicy** selected (indicated by a red box), and a **View** link. At the bottom of the modal, it shows **1 policy selected** and an **Attach** button (also highlighted with a red box).

11. Once you get to this point, you should verify:
- You have a *thing* (Registry -> *Things*).
 - You have a certificate attached to the thing (from the *thing*, click on Security).
 - The certificate is Active (click on the Certificate and look for “Active” in the upper left).
 - The certificate has a policy attached to it (from the Certificate, click on Policies).
 - The policy allows all IoT actions (`iot:*`) for any resource (`*`) (click on the Policy).

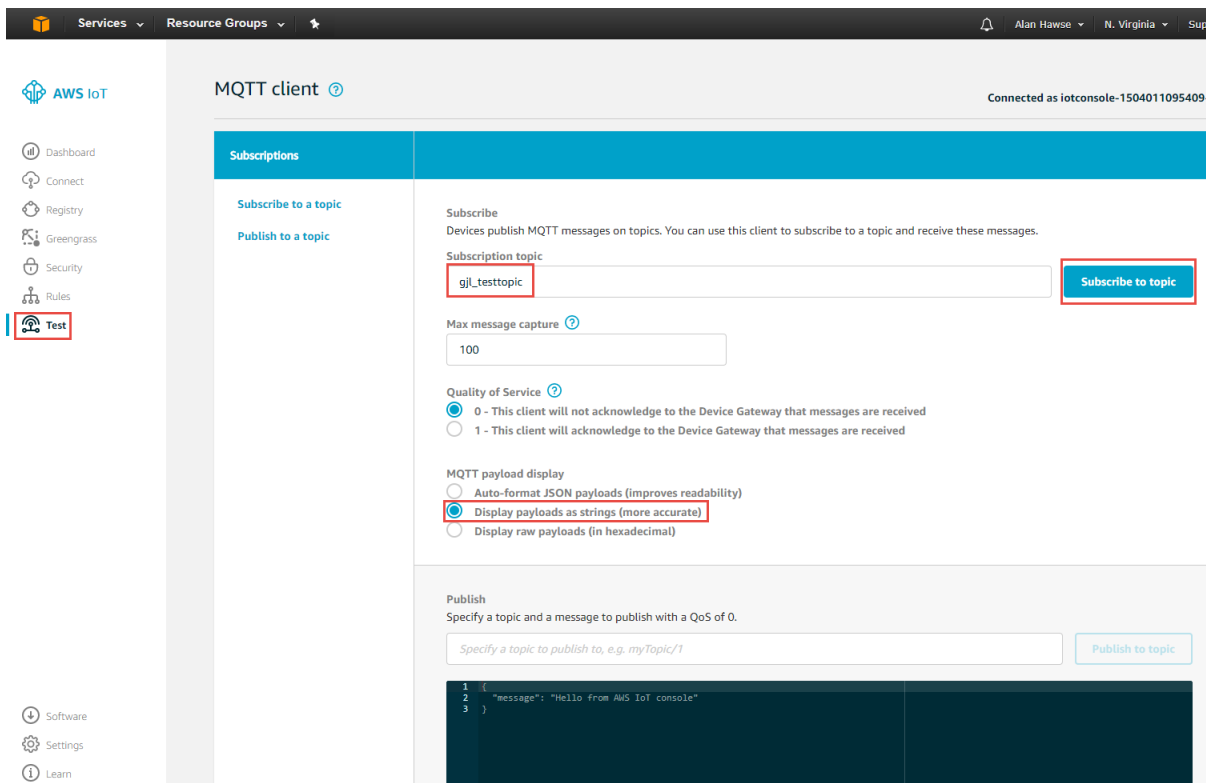
If any of the above is not true, fix it before proceeding. Most of this can be accomplished from the “Actions” menus in the appropriate page. Ask for help from an instructor if you need it.

Exercise - 7B.3 Learn how to use the AWS MQTT Test Client

The AWS website has an MQTT Test Client that you can use to test publishing and subscribing to topics. Think of it as a terminal window into your message broker, or as a generic IoT *thing* that can publish and subscribe. You will use this client to test the later exercises.

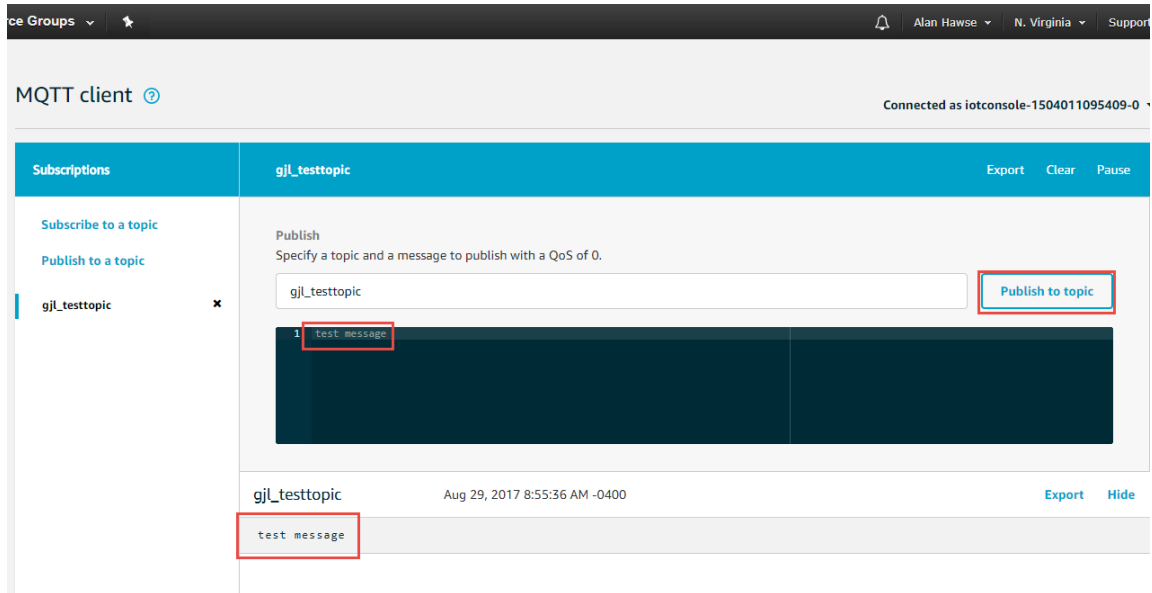
To use the client:

1. Select “Test” from the panel on the left of the screen. Enter a topic that you want to subscribe to such as “<your_initials>_testtopic”, select “Display payloads as strings”, and click on “Subscribe to topic”. Make sure to put your initials or some other unique string in the topic if you are using the class AWS account. If not, you may see messages from someone else publishing to the same topic.



The screenshot shows the AWS IoT console's MQTT client interface. On the left sidebar, the 'Test' option is highlighted. The main panel is titled 'MQTT client' and shows the 'Subscriptions' tab. The 'Subscribe to a topic' section is active. The 'Subscription topic' field contains 'gjl_testtopic'. The 'Max message capture' is set to 100. Under 'Quality of Service', the '0' option is selected. Under 'MQTT payload display', the 'Display payloads as strings (more accurate)' option is selected. The 'Publish' section at the bottom has a 'Publish to topic' button. A message history box at the bottom shows a received message: { "message": "Hello from AWS IoT console" }.

- Now that I am subscribed to a topic I can publish messages to that topic from the MQTT test client. To do this fill in the name of the topic as “<your_initials>_testtopic”. Then type in your message and press “Publish to topic”. You can see in the box below I sent “test message”.



The screenshot shows the MQTT client interface. On the left, under 'Subscriptions', the topic 'gjl_testtopic' is listed. The main panel shows the 'Publish' section with the topic 'gjl_testtopic' entered in the input field and the 'Publish to topic' button highlighted with a red box. Below the input field, a message 'test message' is shown in a dark box, also highlighted with a red box. At the bottom, a message log shows the topic 'gjl_testtopic' with the message 'test message' and the timestamp 'Aug 29, 2017 8:55:36 AM -0400'. The message 'test message' in the log is also highlighted with a red box.

7B.6 References

Resources	Link
AWS Developers Guide	http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html
AWS IOT Getting Started	https://aws.amazon.com/iot/getting-started/
A nice powerpoint about MQTT	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
Avnet Getting Started	http://cloudconnectkits.org/system/files/GSG-BCM4343W%20IoT%20Starter%20Kit%20-%20Getting%20Started%20%28v1.1%29.pdf
Avnet User Guide Part1 and Part2	
AWS Forum	https://forums.aws.amazon.com/forum.jspa?forumID=210