Chapter 7C: Amazon Web Services (AWS)

Time 2 ½ Hours

At this end of Chapter 7C 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.)

How to write WICED firmware to interact with the AWS IoT Cloud using MQTT

7C.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.

7C.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.
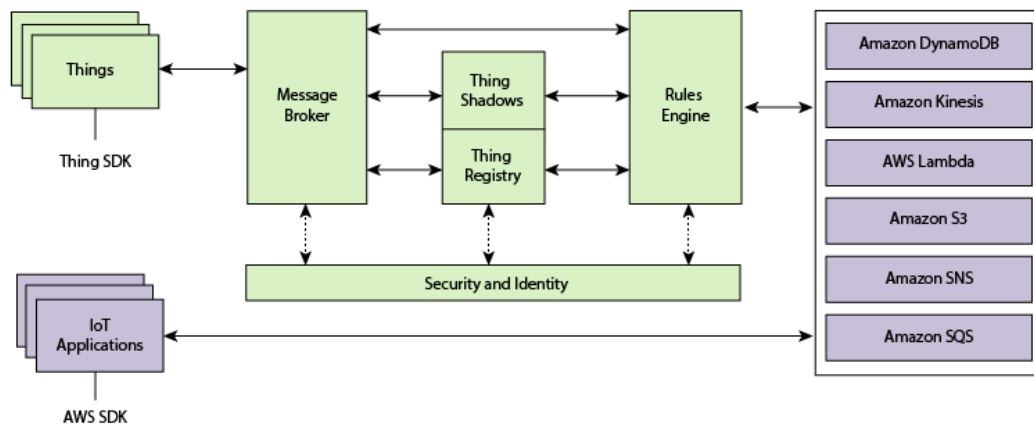
7C.3   AWS IoT Resources

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"
      }
    ]
}
```
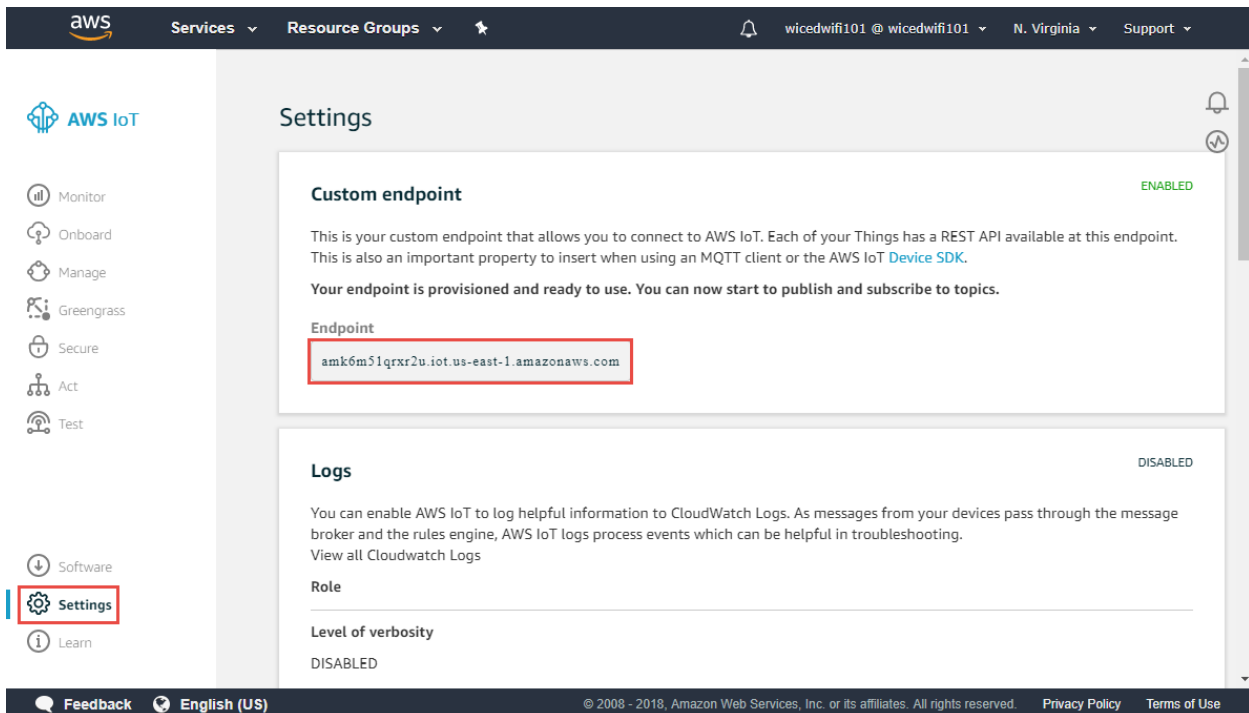
7C.4 AWS IoT Console

7C.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 (called an IAM account instead of a root account) that you can use for the exercises. However, the password for that account may 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.

7C.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"}}}
```

7C.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.

7C.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 *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.

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.

7C.5 Using MQTT with AWS

The WICED SDK contains a library of functions that make it easier to create MQTT firmware. These functions are contained in libraries/protocols/MQTT. To include the library in a project, two things are needed:

In the Make File: $(NAME)_COMPONENTS := protocols/MQTT

In the C Source: #include "mqtt_api.h"

In addition to the library, there are several demo applications that can be used as a starting point for using MQTT with AWS. These applications are: apps/demo/aws_iot:

- *apps/demo/aws_iot/pub_sub/publisher*
    - o   Publish a message when a button is pressed on the kit.
- *apps/demo/aws_iot/pub_sub/subscriber*
    - o   Subscribe to messages that control an LED on the kit.
- *apps/demo/aws_iot/shadow*
    - o   Interact with a *thing* via the shadow. This project also demonstrates using a configuration AP and a web server to set up the Wi-Fi configuration and security certificates.
- *apps/demo/aws_iot/temperature_controlled_device/device*
    - o   Subscribe to messages from the remote sensor and turn an LED ON/OFF depending on the messages received. This project also uses a configuration AP and web server to set up the Wi-Fi configuration and security certificates.
- *apps/demo/aws_iot/temperature_controlled_device/remote_sensor*
    - o   Measure room temperature from a thermistor and get the outside temperature from openweathermap.org. Depending on the temperature difference, publish messages to the temperature controlled device. This project also uses a configuration AP and web server to set up the Wi-Fi configuration and security certificates.

The security certificates and keys for each of these projects are read from resources/apps/aws_iot. The keys are included in the project using the following:

In the Make File: $(NAME)_RESOURCES := apps/aws_iot/rootca.cer \
                         apps/aws_iot/client.cer \
                         apps/aws_iot/privkey.cer

In the C Source: #include "resources.h"

In the exercises below, we will use the publisher, subscriber, and shadow projects as starting points to learn the details of using WICED to interact with AWS using MQTT.

7C.6 Using HTTPS with AWS

In addition to MQTT, AWS supports a [REST API](#) interface to their cloud. The REST API Endpoint is:

https://<your_endpoint>:8443/things/<your_thing_name>/shadow

Note that the port that AWS uses for secure HTTP traffic is 8443 instead of the typical 443.

The connection must have a client verified connection (you need to provide your certificate and private key. Use wiced_tls_init_identity to initialize the identity and then provide a pointer to the identity to the http_client_init function. After you have a connection you can GET, POST and DELETE the document which is in JSON format.

Here is an example of a CURL connection to AWS:

```
CURL -v --cert 6fb5d874d6-certificate.pem --key 6fb5d874d6-private.pem --cacert rootca.cer -X GET
https://amk6m51qrxr2u.iot.us-east-1.amazonaws.com:8443/things/ww101_39/shadow
```

7C.7 Exercise(s)

Exercise - 7C.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 account. The login for the class account is:

Account Alias: wicedwifi101
IAM User Name: wicedwifi101
Password: See the back cover of manual for the current password or ask an instructor
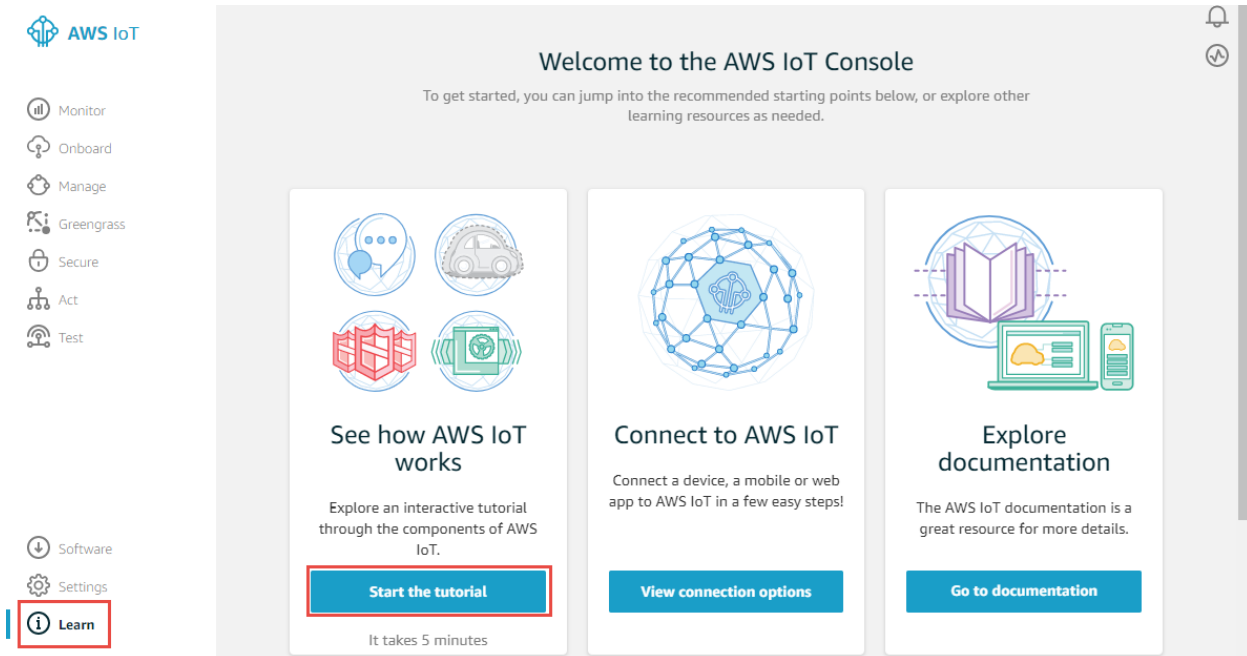
If you use the class account, leave the region set to "N. Virginia". If you create your own account, you can use any region you want but the exercises assume N. Virginia.
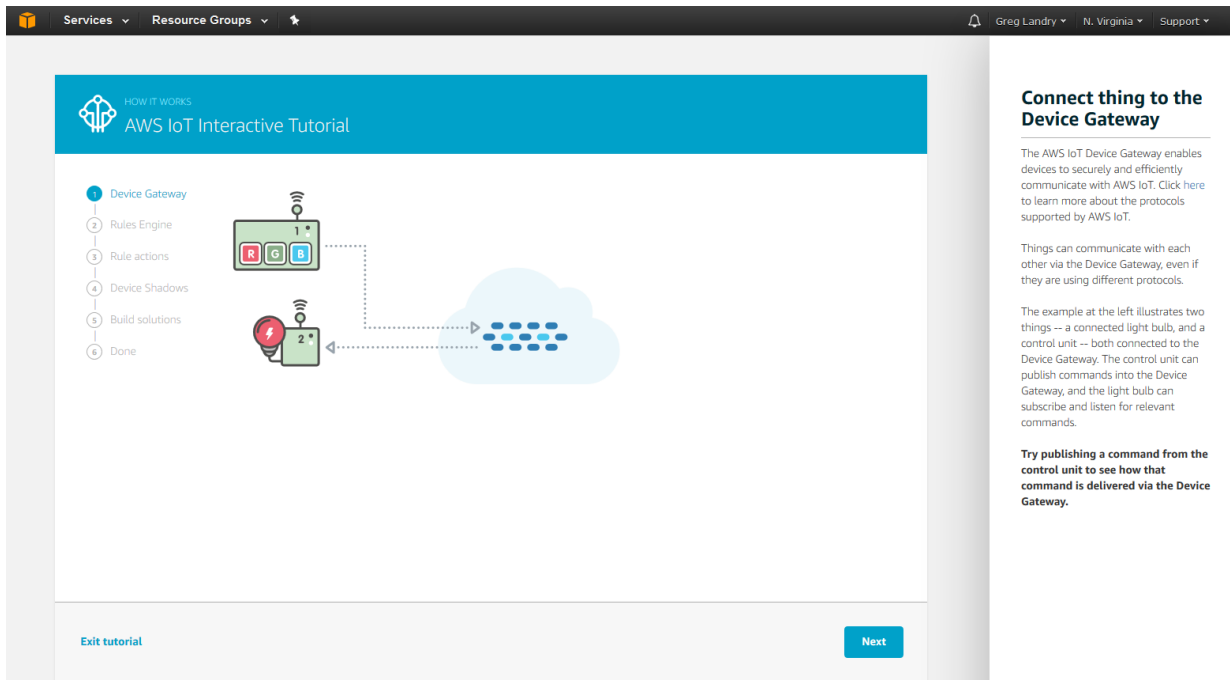
Once you are logged in, from the Services menu, select "AWS IoT":
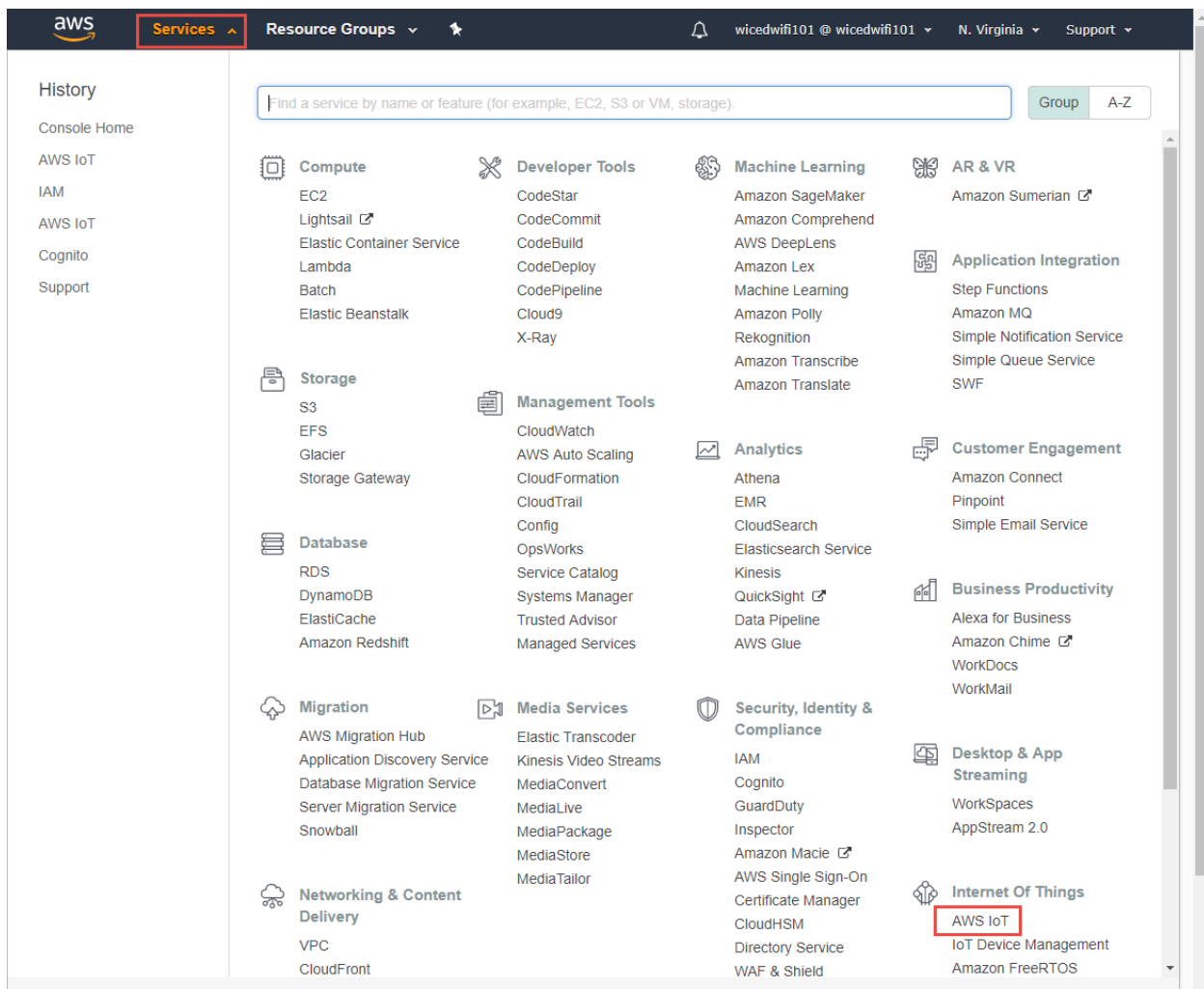
In the lower-left corner of the IoT screen click on "Learn" and then click "Start the tutorial":

Follow the instructions to complete the tutorial.
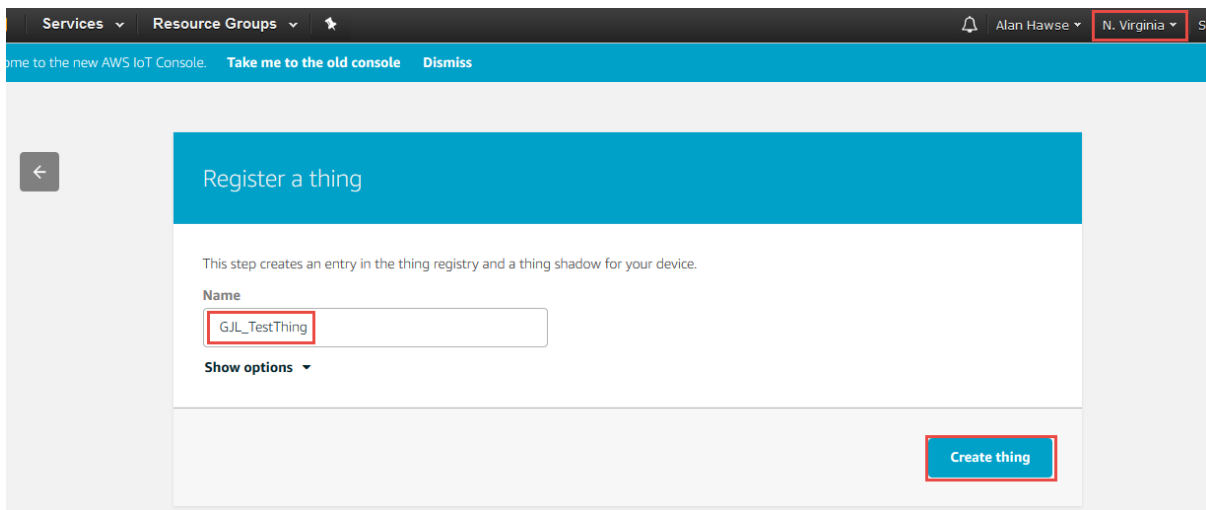
Exercise - 7C.2 Create new AWS Thing
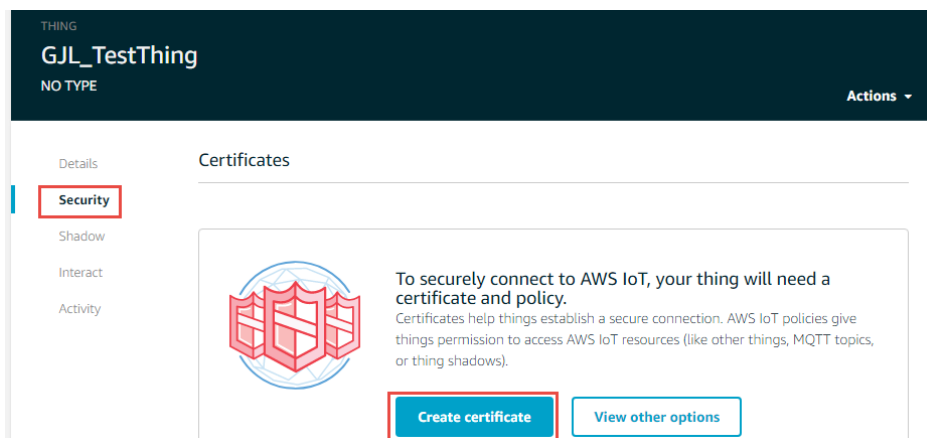
Provision a new *thing* in the AWS IOT Cloud, and establish its policy and credentials.

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.

1. 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 log into the class account make sure it is set to that region when you log in. 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.
2. Name your *thing* **<YourInitials>_TestThing** (or whatever) and press "Create thing".

3. 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, find your *thing* in the list of *things* and click on it. If you don't see it in the list, you can search for it using the search box at the upper right corner of the window. One you get to your *thing's* 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 (the long string in the filenames that you downloaded) since you will 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.
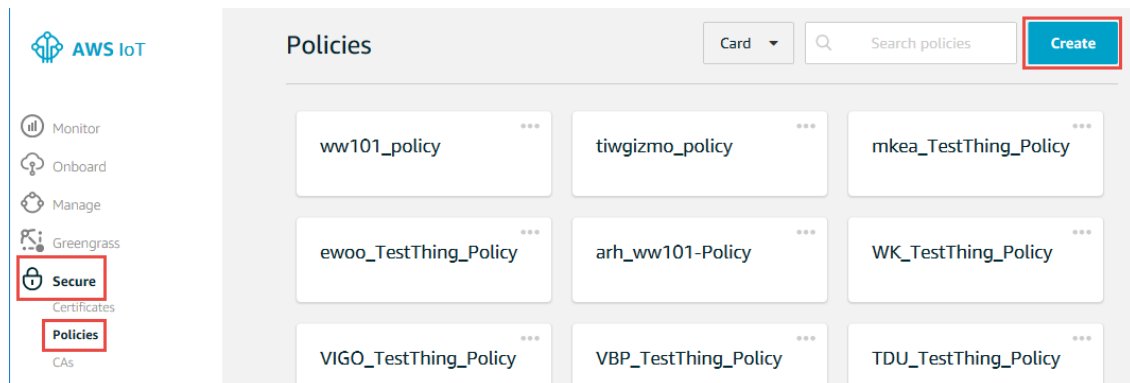
6. Click "Done" once you have downloaded the files and have noted down your certificate ID. This will take you back to your *thing's* page. Click the left arrow at the upper left-hand corner to go back to the top-level AWS IoT page. From that page click on "Secure", then "Policies", and finally on "Create".

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.
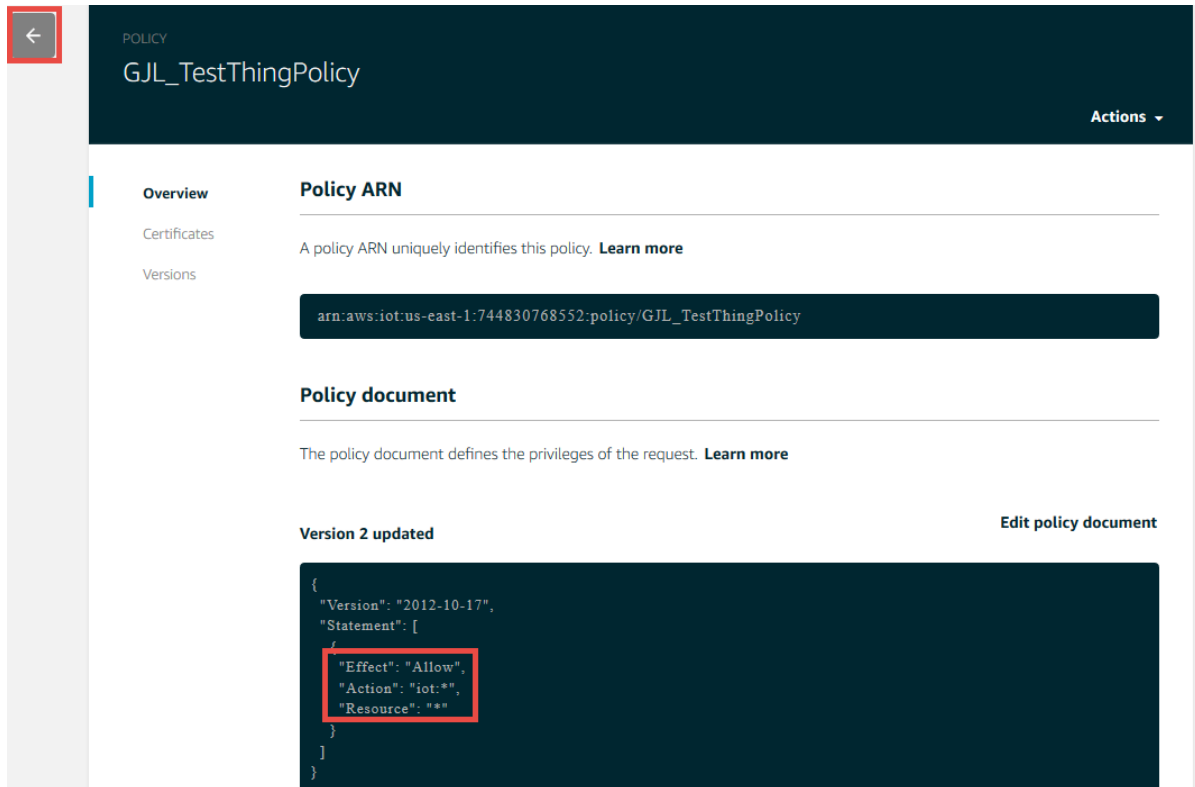
8. If you click on the policy, you will see the policy document details. In this case, any IoT operation (iot:*) is allowed for any resource (*).
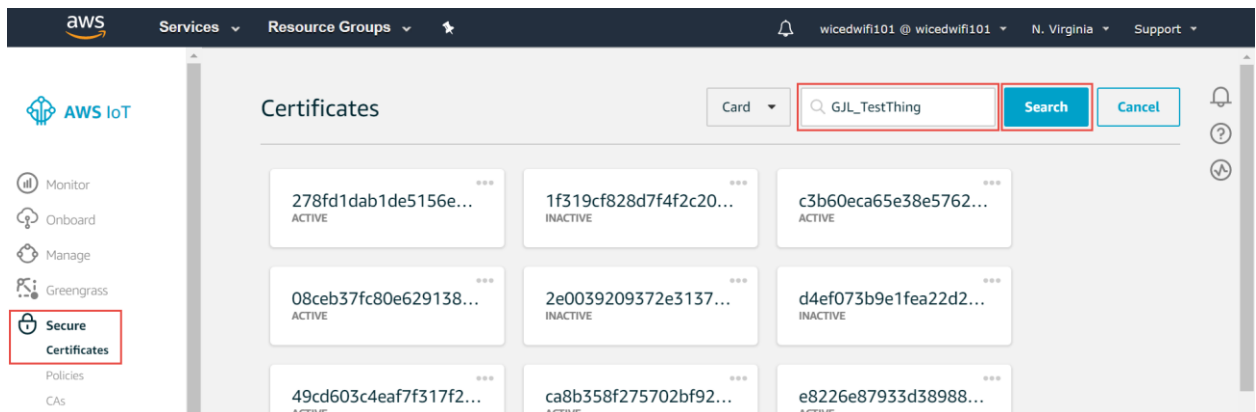
9. You now need to attach the policy to the certificate. First click the left arrow on the left side of the screen as show above. Then select Secure -> Certificates from the left panel, and click on your certificate.
Again, you can use the search box in the upper right corner to find your certificate 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.
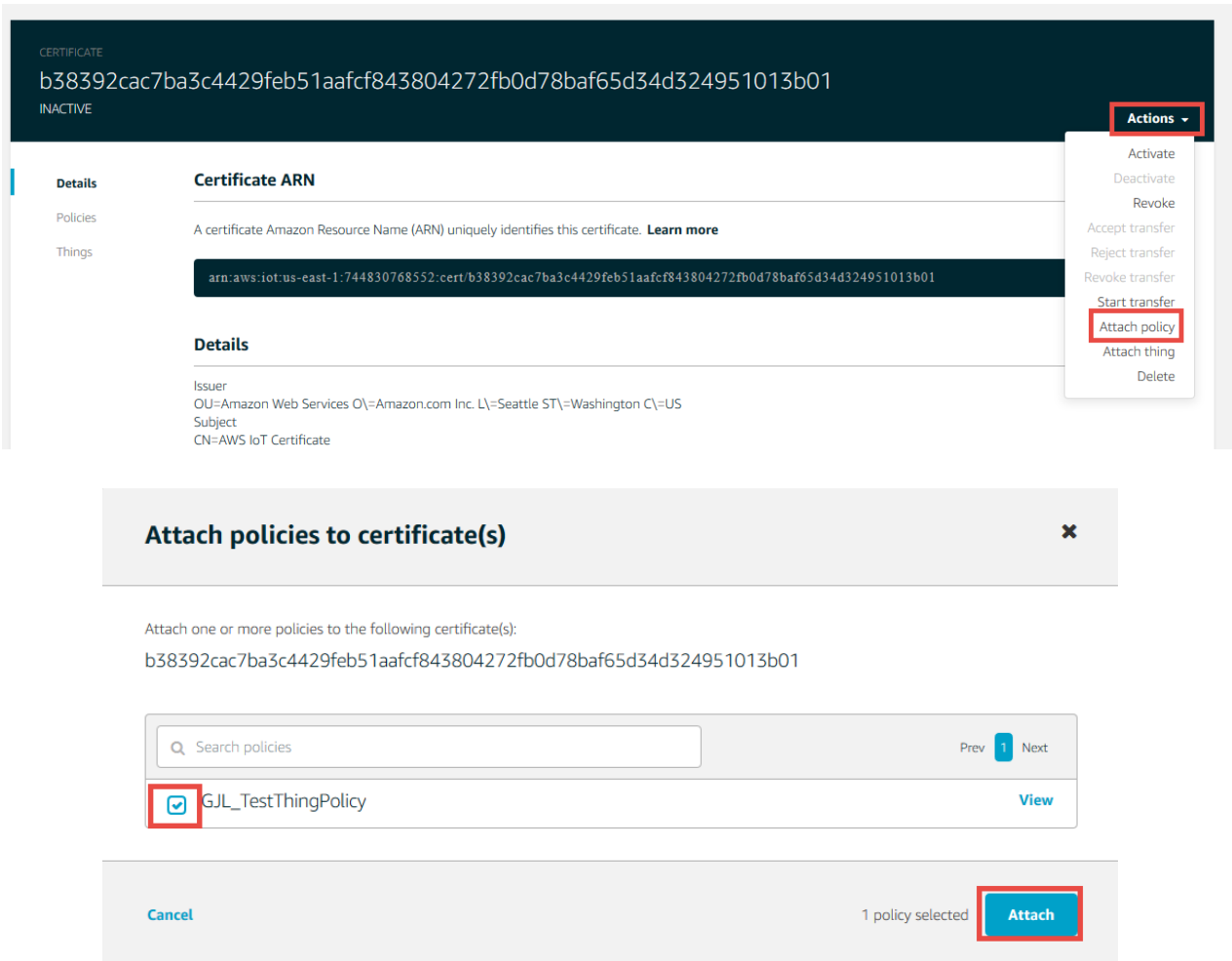
10. Once you click on your certificate, select "Actions -> Attach Policy". Select your policy and click "Attach". Click on the left arrow in the upper left when you are done to return to the AWS IoT main page.

11. Once you get to this point, you should verify:
    a. You have a *thing* (Manage -> *Things*).
    b. You have a certificate attached to the thing (from the *thing*, click on Security).
    c. The certificate is Active (click on the Certificate and look for "Active" in the upper left).
    d. The certificate has a policy attached to it (from the Certificate, click on Policies).
    e. 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.
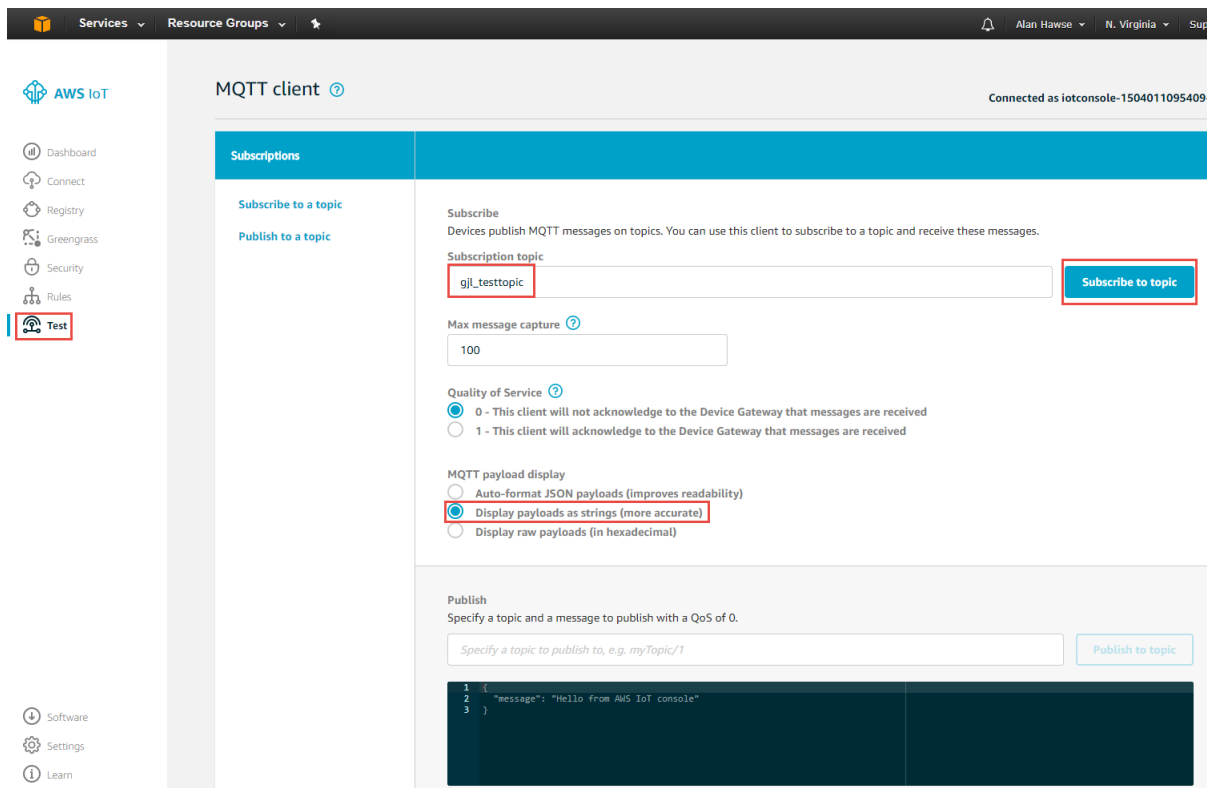
Exercise - 7C.3 Learn how to use the AWS MQTT Test Client

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 make the subscribe/publish actions more understandable, it is useful to team up with another student for this exercise. Alternately, you can run two tabs in your browser – one to subscribe and one to publish.

Subscribing to a Topic from the Test Client:
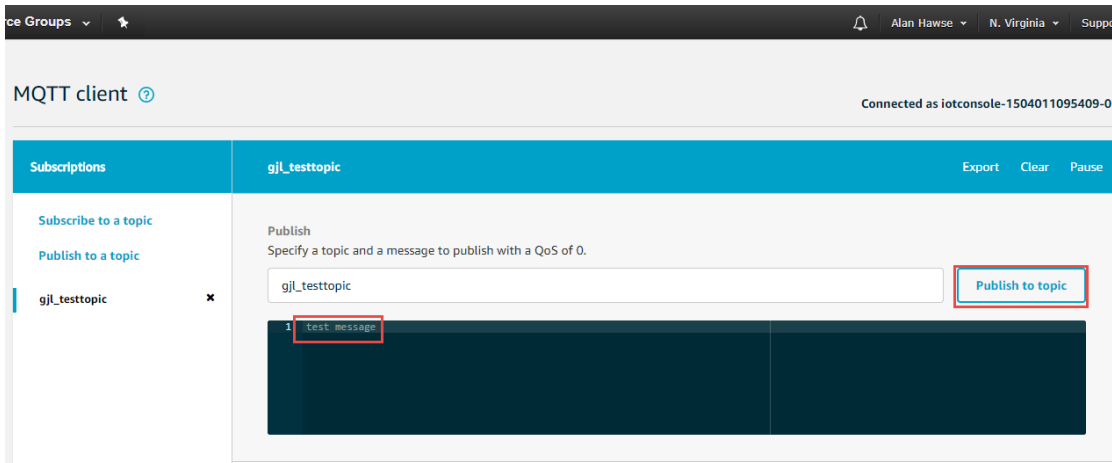
1. Select "Test" from the panel on the left of the screen. Enter a topic that you want to subscribe to such as **<your_initials>_testtopic** in the "Subscription topic" box. Select "Display payloads as strings", and click on "Subscribe to topic". Make sure to put your initials or some other unique string 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.

Publishing to a Topic from the Test Client:

2. Now that I am subscribed to a topic I can publish messages to that topic from another instance of the MQTT test client. First, open another web browser tab, login to the AWS account, and go to the Test page. (Note: team up with another student if you can so that one person subscribes and the other publishes to the same topic).

3. Scroll down to the "Publish" section of the page and fill in the name of the topic that you subscribed to earlier. The name must be exactly the same (including case). Then type in your message and press "Publish to topic". You can see in the box below I sent "test message".
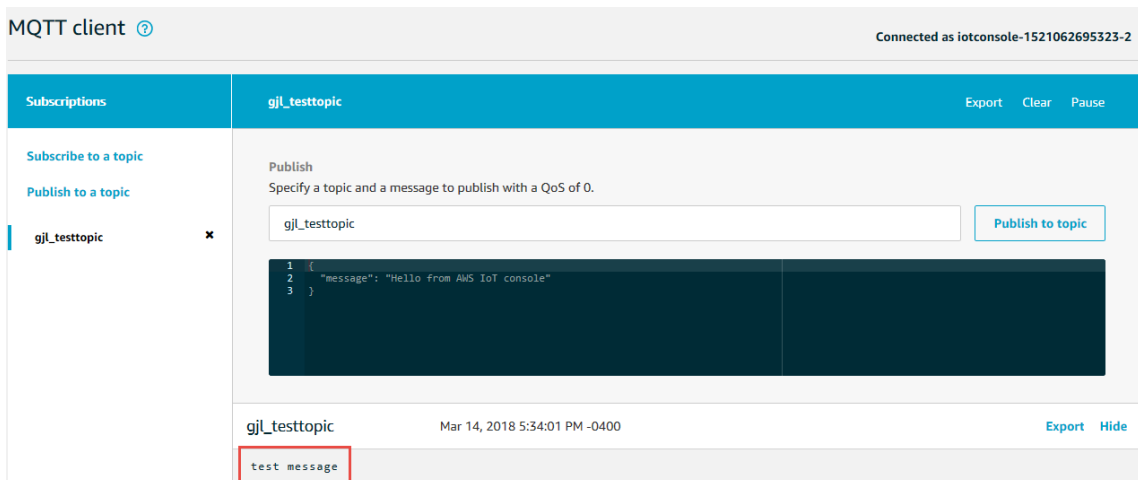
4. Now, go back to the tab with the subscription and see that the published message was send to the subscriber.

Exercise - 7C.4 Run 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. ww101/07c/04_publisher) and update the files.
    a. Hint: Make sure you add your platform to the valid platforms in the makefile.
2. Modify the DCT for your network.
3. Copy the certificates that you generated in the first exercise into the resources/apps/aws_iot directory. Replace two of the existing files in that directory as follows:

Name of Downloaded File	New Name	Description
<name>-certificate.pem.crt	client.cer	The certificate for your thing. This is how AWS knows that it is a valid *thing* that is trying to talk to it.
<name>-private.pem.key	privkey.cer	The private key that your application will use to decrypt data that it gets back from AWS. Since Amazon created the key, it already has the public key.

Make sure you change the file extensions to match what is shown above. Windows explorer will hide some file extensions, so it is best to look at them inside the WICED Studio Project Explorer to make sure they are right.

The rootca.cer file in that folder is the certificate for Amazon. This allows your thing to know that it is really talking to the AWS Cloud. This is a known-good key for AWS that is built into the SDK. It does not need to be modified since it never changes (at least not until it expires in on July 16, 2036).

The other file that you downloaded called "<name>-public.pem.key" is a public key for your thing. In this case, Amazon already has the public key, so you don't need to provide it.

4. Run a "clean" before rebuilding or else your project may not see the new keys. You will find clean at the top of the list of Make Targets. Just double-click on it to run it.
5. Create a Make Target for your project.
6. Modify the #define for MQTT_BROKER_ADDRESS. The correct address can be found by clicking on "Settings" at the lower left corner of the main Amazon AWS console window. The broker address is listed as the "Endpoint".
7. If you are not using the N. Virginia region, replace us-east-1 in the MQTT_BROKER_PEER_COMMON_NAME with your region. Note that the region is part of the Endpoint string that you entered in the previous step. Make sure the region matches for both strings.
8. Modify the #define for WICED_TOPIC. You can use the topic from the previous exercise with your initials in the name.
9. Modify the #define for CLIENT_ID to include your initials. This is necessary to prevent conflicts between multiple devices – every device MUST have a unique Client ID. In a production product, it would probably be a good idea to use the device's MAC address as part of the Client ID string since that guarantees a unique value for every device.

  a. Hint: The solution project includes the MAC address in the Client ID. Search for "WW101" in the solution project to see the changes.

10. Build and program your project.

11. Open the serial port and watch your terminal session.

12. Subscribe to your topic using the AWS MQTT Test Client. When you press the button, you should see updates to the topic in the test window.

Exercise - 7C.5 WICED MQTT Firmware Flow

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

1. How do the MQTT library functions (e.g. *wiced_mqtt_publish()*) get into your project?

2. What function is called when the button is pressed?

3. How does the button callback unlock the main thread?

4. What WICED SDK RTOS mechanism does the "wait_for_response" function use to "wait"?

5. Why did the firmware author create a function called "wait_for_response"?

6. Are all messages sent to the AWS IOT MQTT Message Broker required to be in JSON format?

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

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

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

10. What steps are required to get an MQTT connection established?

11. What prevents a hung connection from deadlocking the publisher app?

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

Exercise - 7C.6 Build and Run 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. wa101/07c/06_subscriber) and modify the DCT and makefile.
2. Update the topic and broker #defines to the same ones you used for exercise 4.
3. Update the #define for CLIENT_ID to contain your initials so that it is a unique value or use your device's MAC address.
    a. Hint: The solution project includes the MAC address in the Client ID. Search for "WW101" in the solution project to see the changes.
4. We will use the same *thing*, certificate, and keys that we did for exercise 2.
5. Publish messages using the AWS Test MQTT Client.
    a. Determine what string needs to be sent to turn the light on or off.
        i. Hint: Look in the source code to find the string that is being looked for when a message is received.
        ii. Hint: If you are successful, the LED on the shield should turn on/off.

Exercise - 7C.7 (Advanced) Implement the publisher and subscriber in two different kits

1. In a real-world application, you would typically have one or more devices publishing data to a broker and one or more devices subscribing to updates from that same broker. So, let's try that out with two different kits. You should team up with another student for this lab.
    a. Have one student use the subscriber project and the other student use the publisher project.
    b. Use the same Topic Name the same in both projects.
    c. The two projects MUST have different CLIENT IDs or it will not work – the two kits will interfere with each other.
        i. The publisher and subscriber solution projects include the MAC address in the Client ID to ensure that it is unique for each kit. Search for "WW101" in the solution projects to see the changes.
    d. Program the updated firmware into the two kits – one publisher and one subscriber.
    e. Power up both kits.
    f. Subscribe to the topic that you chose using the AWS test MQTT Client.
    g. Press the button on the publisher kit and watch it change the state of the LED on the subscriber kit. Also watch the messages in the AWS test MQTT Client window.

Exercise - 7C.8 (Advanced) Build and test the Shadow App

1. This example uses a configuration Access Point serving a web browser so that the *thing* name, credentials, keys, and settings for the network to connect to can be configured from a web browser on a device attached to the configuration AP.
2. Copy the WICED application from *apps/demo/aws_iot/shadow* to your directory and update the makefile.
   Update the DCT file to have a Config AP for configuration with an SSID name that is unique (so as not to collide with others in your class).

3. Update the message broker address to match what you created in the previous exercises.
    a. Hint: The message broker address goes in "*aws_common.h"* in the #define for AWS_IOT_HOST_NAME.
    b. Make sure you also update the region in AWS_IOT_PEER_COMMON_NAME if you are not using us-east-1.
4. Update the #define for CLIENT_ID to contain your initials so that it is a unique value or use your device's MAC address.
    a. Hint: The CLIENT_ID is in the file *aws_common.c.*
    b. Hint: The solution project includes the MAC address in the Client ID. Search for "WW101" in the solution project's *aws_common.c* file to see the changes.
5. Program the kit.
6. Attach to the Config AP on your board from your computer's Wi-Fi.
    a. Connect to the SSID that you programmed into the board.
    b. Go to the webserver (The IP address is printed on the terminal when the device boots and starts the AP).
        i. Hint: Don't use Firefox for this step – it sometimes gives strange results.
    c. Update the *thing* name to match what you created in previous exercises.
        i. Hint: The default name that shows up is in "*aws_config.h"* so you could also change it there before programming the board.
    d. Follow the instructions to upload the client certificate and private key.
    e. Click on "Wi-Fi Setup >", click on the class Wi-Fi network, enter the password, and click connect.
    f. The board will reboot. Once it has done that, it will connect as a station to the Wi-Fi network that you configured in the previous step.
7. Attach to a Wi-Fi access point from your computer.
8. Go to console.aws.amazon.com and go to the test MQTT Client.
9. Subscribe to the device's shadow topics.
    a. Hint: *$aws/things/<YourThingName>/shadow/#* will subscribe to all shadow topics for your *thing*. Replace *<YourThingName>* with the name that you chose for your *thing*. The # is a wildcard.
10. Press the button on the board and see the messages.
11. Answer the question: What is the sequence of events that changes the LED from On to Off?

Exercise - 7C.9 (Advanced) Get a *Thing* Shadow from AWS using HTTPS

1. Create a project that uses HTTPS to connect to AWS. Get the shadow of your *thing* and print it to the UART.
   a. Hint: Start with the httpbin.org GET project using TLS from the HTTP chapter.
   b. Hint: In addition to initializing the root certificate for AWS, you will need to read in a valid *thing* certificate and private key to initialize the TLS identity. This is necessary because AWS will validate that your WICED kit is authorized to connect to the broker and interact with your *thing*.
   c. Hint: Remember that only 1 certificate can be stored in the DCT so you will have to use another method for at least 2 of the three files required.

Related Example "Apps"

App Name	Function
demo.aws_iot_pub_sub/publisher	Demonstrates publishing information to the AWS cloud.
demo.aws_iot_pub_sub/subscriber	Demonstrates subscribing to a topic in the AWS cloud.
demo.aws_iot_shaddow	Demonstrates using a shadow device with AWS.

7C.8 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