Chapter 4c: Cloud Protocol: MQTT Using AWS

At this end of this chapter you will understand how to use the MQTT protocol with Amazon Web Services (AWS), including:

- How to write ModusToolbox™ any cloud firmware to interact with the AWS IoT Cloud using MQTT
- How the AWS Cloud works
- 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
- How to use a *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.)

Table of contents

Document conventions

Convention	Usage	Example
Courier New	Displays code and text commands	CY_ISR_PROTO(MyISR); make build
Italics	Displays file names and paths	*sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures	[**Enter**] or [**Ctrl**] [**C**]
Menu > Selection	Represents menu paths	**File > New Project > Clone**
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

4c.1 Introduction

Whether you use ModusToolbox™ any cloud, Amazon FreeRTOS, Mbed, or your own solution, at the heart of it your device will connect and communicate using the AWS IoT Core using MQTT. This chapter will discuss some of the concepts that are important to know when connecting your IoT device to AWS.

## 4c.2 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 (this is a partial list – more is being added by Amazon all the time):

- AWS IoT: A cloud platform that provides Cloud services for IoT devices (the subject of this chapter).
- Amazon Elastic Cloud (EC2): 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 Amazon DynamoDB, 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.

4c.3 AWS IoT Introduction

The AWS IoT Cloud service supports MQTT Message Brokers, HTTP access, **plus** a bunch of server-side functionality that includes (again, this is a partial list because Amazon frequently adds more functionality):

- 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.
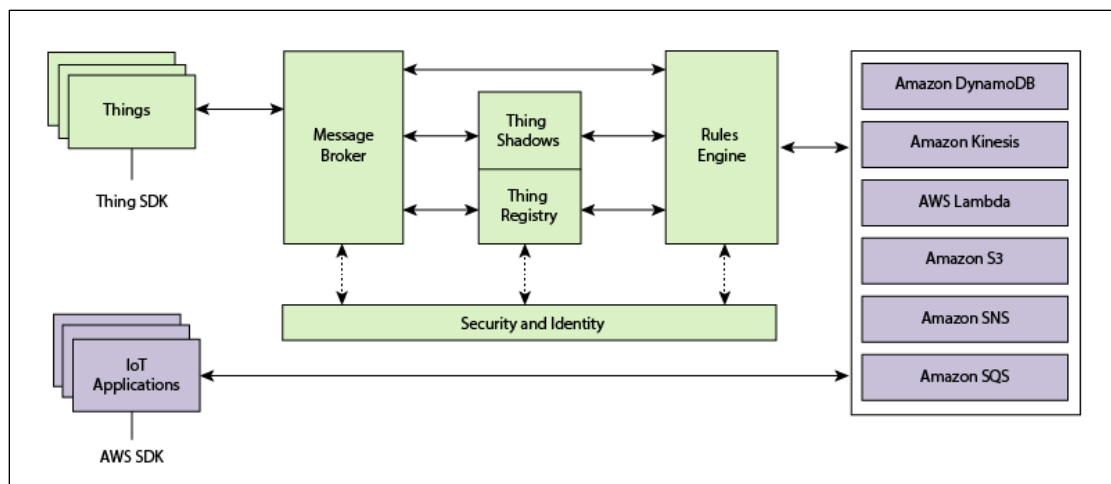
4c.4 AWS IoT Resources

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.

4c.4.1 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).

4c.4.2 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.

4c.4.3 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"
        }
    ]
}
```

4c.5 AWS IoT Console

Note: *Screenshots shown below may not exactly match the AWS site because Amazon updates their site frequently. However, the concepts are the same and you should be able to follow along without too much trouble.*

4c.5.1 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 the year is up. 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 an HTTP server on that machine. To connect your 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 AWS IoT console window. The name is listed as the Endpoint.
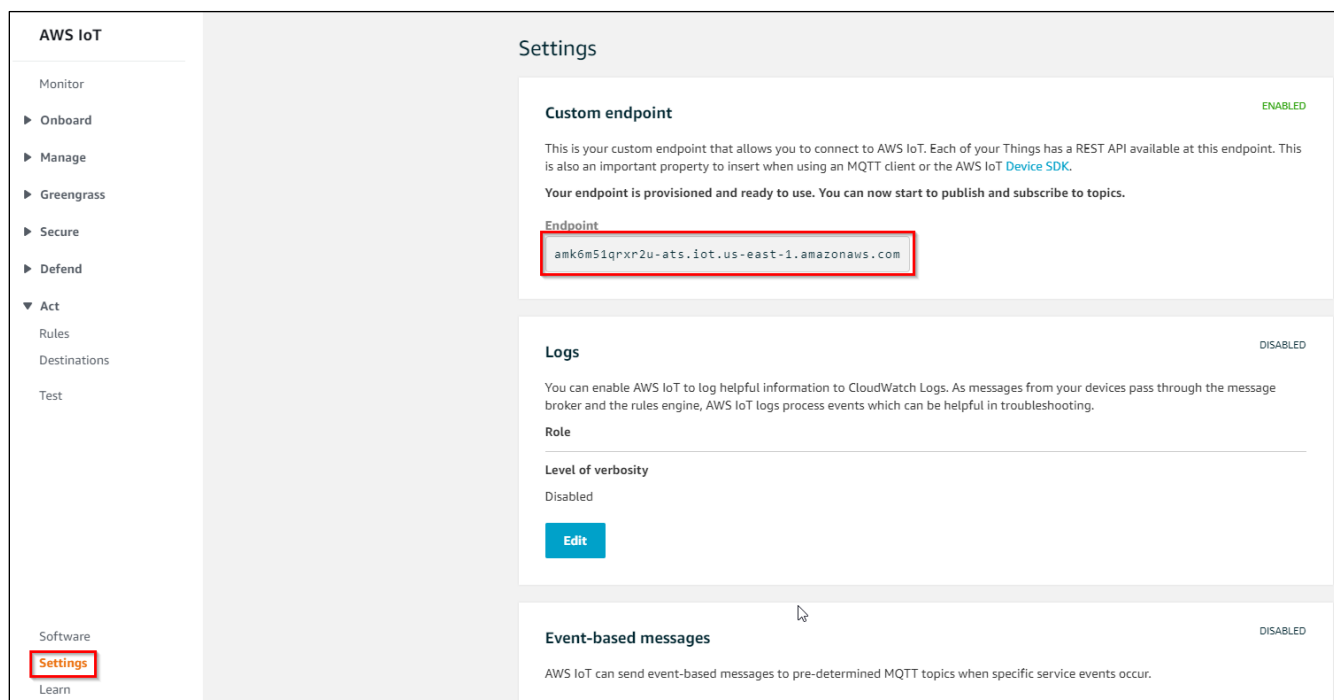
4c.5.2 Thing Shadow

A *thing* shadow (sometimes referred to as a device shadow) is a JSON document (http://docs.aws.amazon.com/iot/latest/developerguide/iot-thing-shadows.html) that is used to store and retrieve current state information for a *thing* (device, app, etc.). 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:

4c.5.2.1 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 it.
- Reported: The reported state of the *thing*. *Things* write to this portion of the document to report their new state. Applications can read this portion of the document to determine the state of a *thing*.

4c.5.2.2 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.

4c.5.2.3 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.

4c.5.2.4 ClientToken:

A string unique to the device that enables you to associate responses with requests in an MQTT environment.

4c.5.2.5 Version:

The document version. Every time the document is updated, this version number is incremented. This is 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" : {
        "desired" : {
            "color": "BLUE"
        }
    }
}
```

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

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

4c.5.3   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.

4c.5.4   Device Shadow Topics

Each *thing* that you have will have a group of topics (https://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-mqtt.html) 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	A JSON message that you publish to this topic will update the 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 entire 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, 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.

MQTT Topic Suffix <type>	Function
/get	If a *thing* publishes a message to this topic, AWS will respond with a message to /get/accepted with the current state of the shadow or to /get/rejected if the operation is not allowed.
/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/documents" to subscribe to updated documents for all *thing* shadows.

4c.6   Using MQTT with AWS

ModusToolbox™ any cloud includes an MQTT client library which uses the [AWS IoT Device SDK MQTT client library](). All features supported by the AWS IoT Device SDK MQTT Library are supported by the ModusToolbox™ any cloud solution. There is a configuration file for this library located in *libs/mqtt/cyport/include/iot_config.h*. You should copy this file to the root directory of your application and use it to adjust default library settings.

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

4c.6.1   MQTT client

In this example, the MQTT client RTOS task establishes a connection with the configured MQTT Broker (AWS IoT Core in this example) and creates two tasks - publisher and subscriber. The publisher task publishes messages on a topic when the user button is pressed. The subscriber task subscribes to the same topic and controls the user LED based on the messages received

In the exercises below, we will use the MQTT client project as a starting point to learn the details of using ModusToolbox™ any cloud to interact with AWS using MQTT.

4c.7 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). 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
```

4c.8 Exercises

Exercise 1: Run the AWS tutorial

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

Note: *Screenshots shown in these exercises may not exactly match the AWS site because Amazon updates their site frequently. However, the concepts are the same and you should be able to follow along without too much trouble.*

1. Sign up for an AWS account.

2. Once you are logged in, from the **Services** menu, select **IoT Core**:
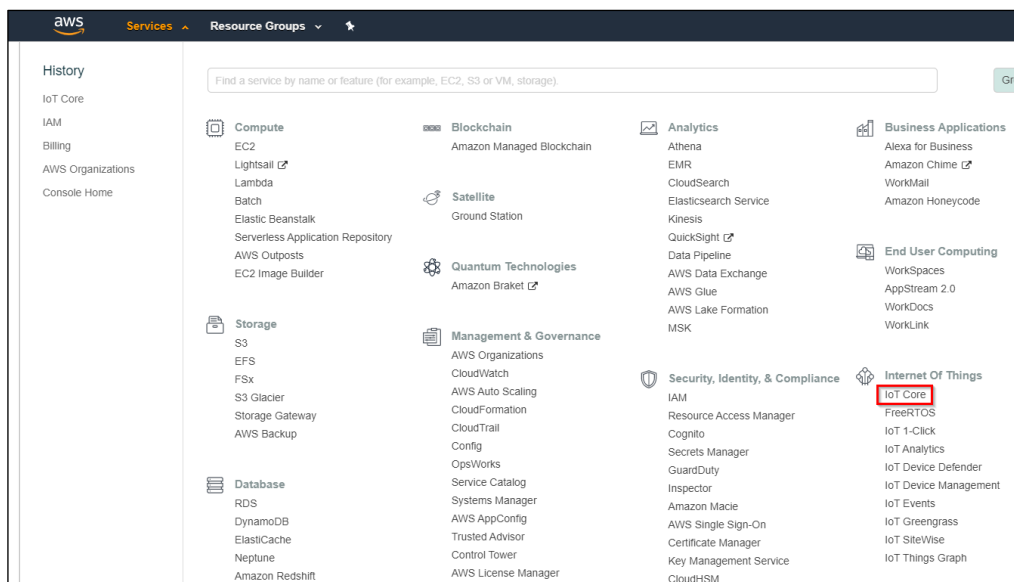
3. In the lower-left corner of the IoT screen click on **Learn** and then click **Start the tutorial**:
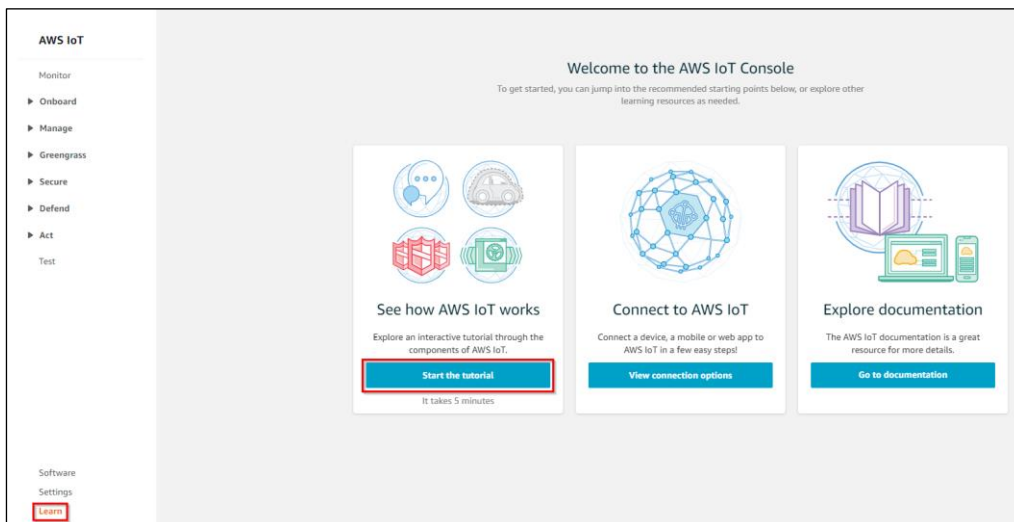
4. Follow the instructions to complete the tutorial.
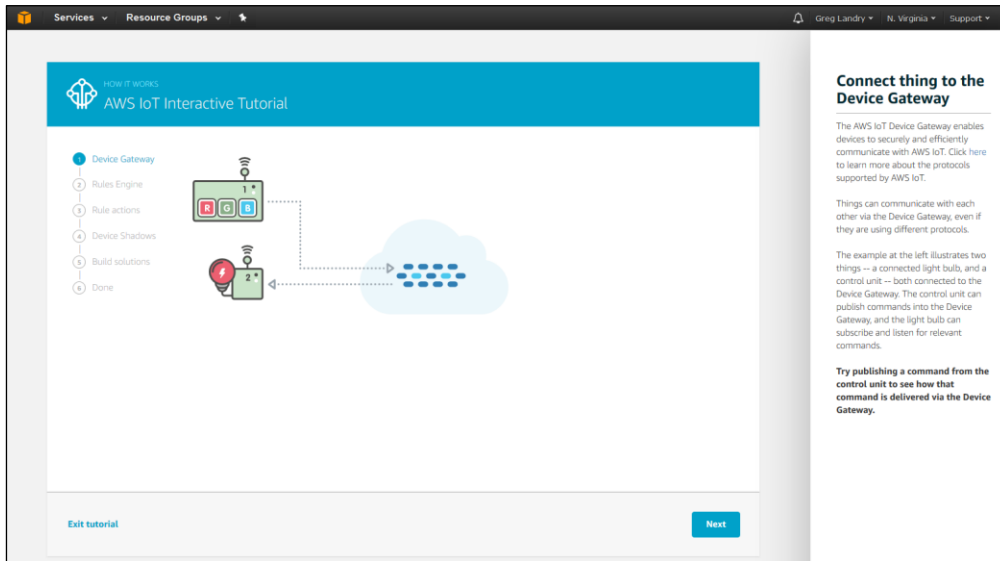
Exercise 2: Create new AWS *Thing*

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

1. Once you have watched the tutorial, you should be on the **Register a thing** page.

2. Name your *thing* **<YourInitials>_TestThing** (or whatever) and press **Create thing**.
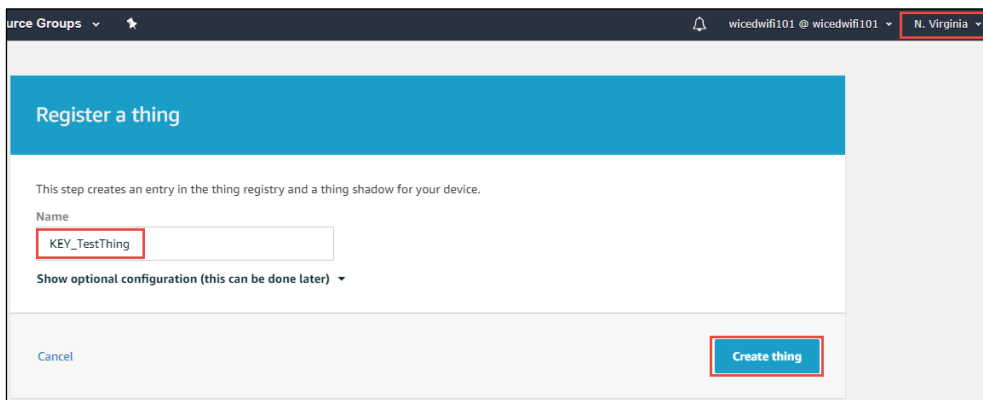
3. Before you can access the broker from your kit you need to create the encryption keys that enable you to identify it as an allowed device.

   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.
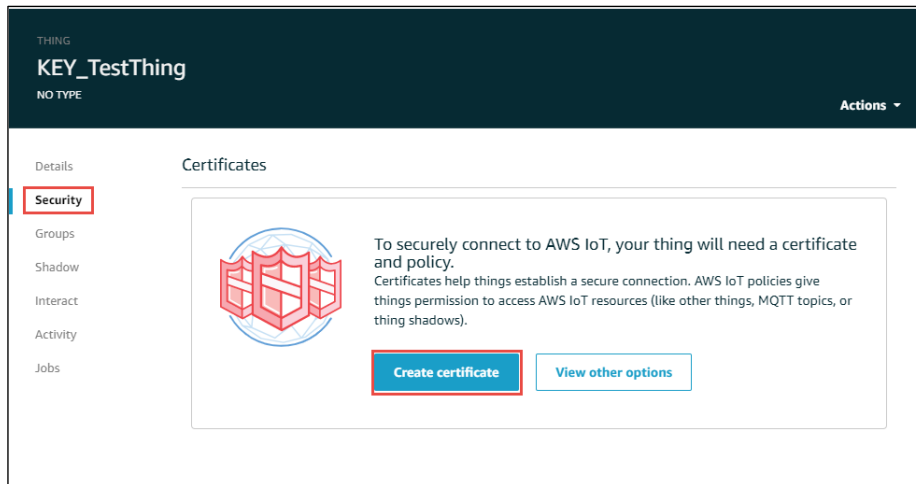
4. Now you need to download the "certificate", "public key" and "private key".

   If you don't download the certificates at this step, you **cannot** come back. So, you must download those files now to make the TLS work!
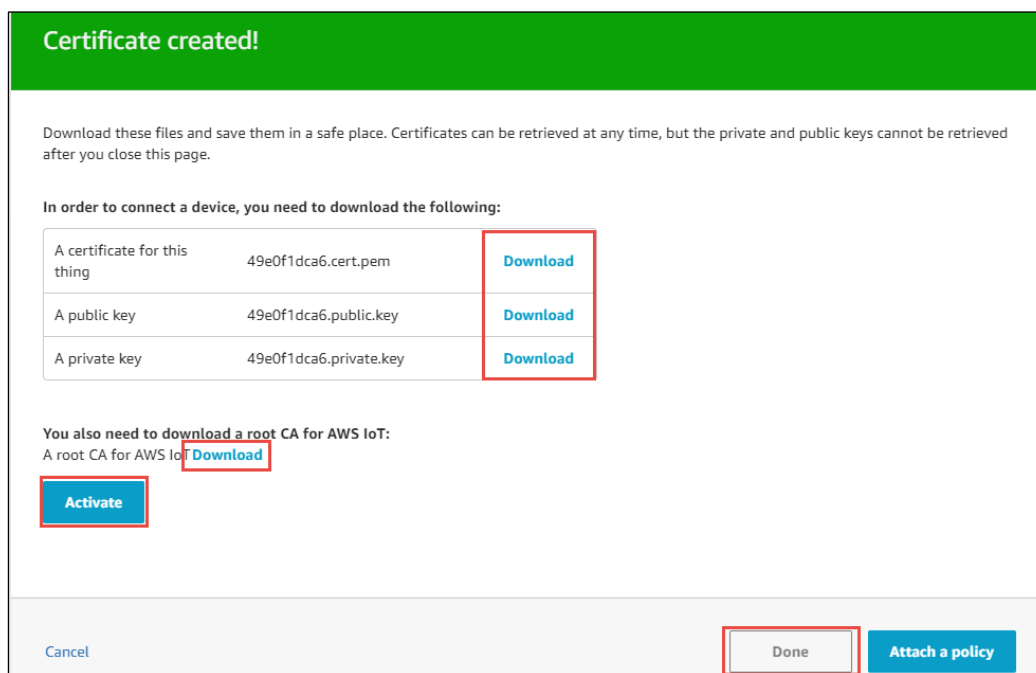
5. Write down the certificate ID (the long string in the filenames that you downloaded) since you will need it later when you attach a policy to the certificate.

6. Activate your certificate by clicking on the **Activate** button.

7. Download the AWS IoT root CA certificate by clicking the Download link.

   This will open a new page with different certificates as shown below. You want to download the "Amazon Root CA 1" certificate by right clicking on it and selecting **Save link as...**

CA certificates for server authentication

Depending on which type of data endpoint you are using and which cipher suite you have negotiated, AWS IoT Core server authentication certificates are signed by one of the following root CA certificates:

VeriSign Endpoints (legacy)

- RSA 2048 bit key: VeriSign Class 3 Public Primary G5 root CA certificate ⧉

Amazon Trust Services Endpoints (preferred)

ⓘ **Note**

You might need to right click these links and select **Save link as...** to save these certificates as files.
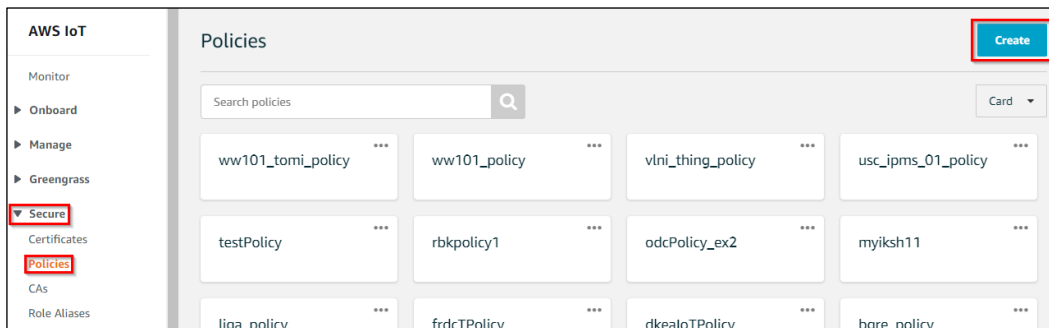
- RSA 2048 bit key: Amazon Root CA 1 ⧉.
- RSA 4096 bit key: Amazon Root CA 2. Reserved for future use.
- ECC 256 bit key: Amazon Root CA 3 ⧉.
- ECC 384 bit key: Amazon Root CA 4. Reserved for future use.

These certificates are all cross-signed by the Starfield Root CA Certificate ⧉. All new AWS IoT Core regions, beginning with the May 9, 2018 launch of AWS IoT
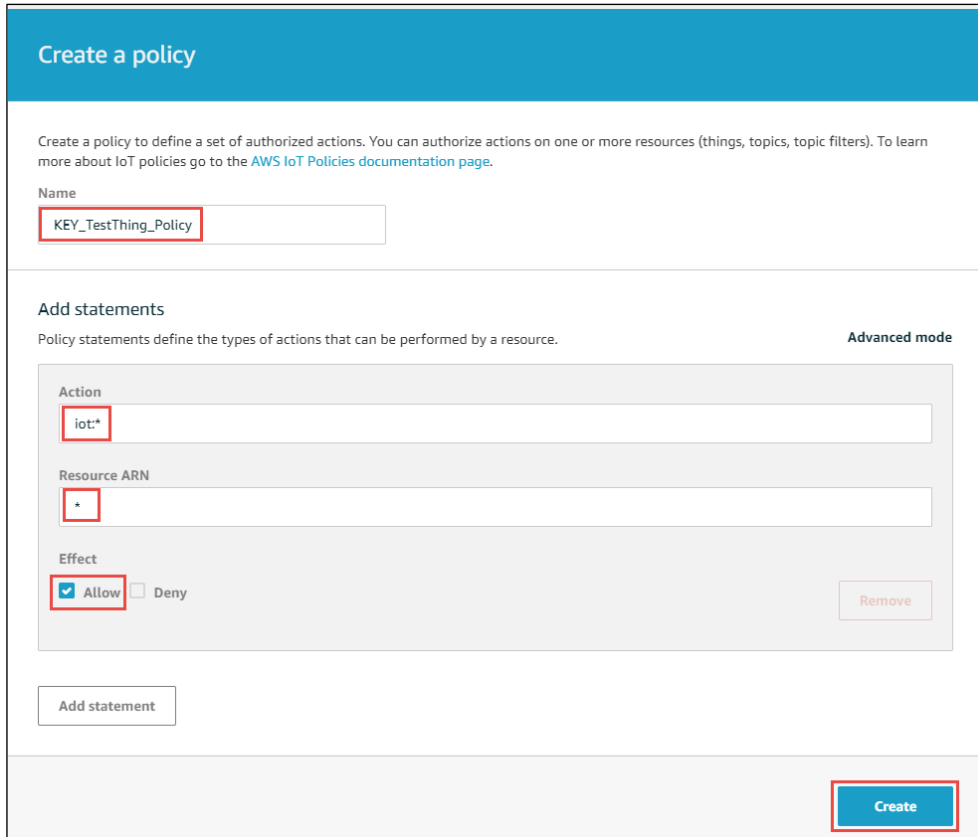
8. Click **Done** <u>after</u> you have downloaded the files, activated your certificate noted down your certificate ID.

   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**.

9. 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.
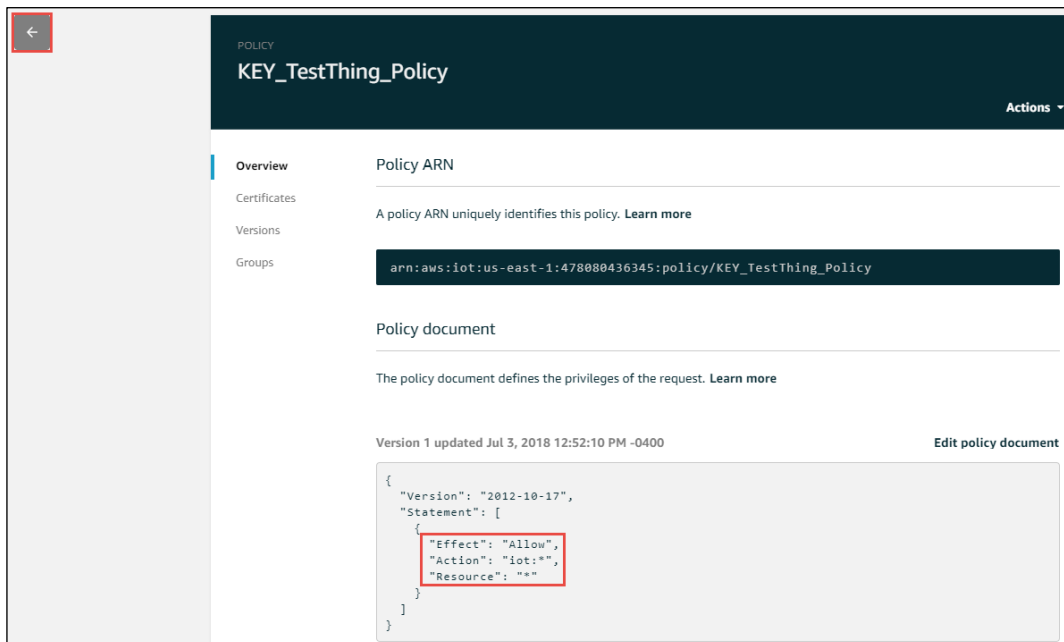
10. If you click on the policy, you will see the policy document details. In this case, any IoT operation (iot:*) is allowed for any resource (*).


11. You now need to attach the policy to the certificate.

a. First click the left arrow on the left side of the screen as show above.

b. Then select **Secure > Certificates** from the left panel and click on your certificate.

c. Again, you can use the search box in the upper right corner to find your certificate by name.

d. 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.
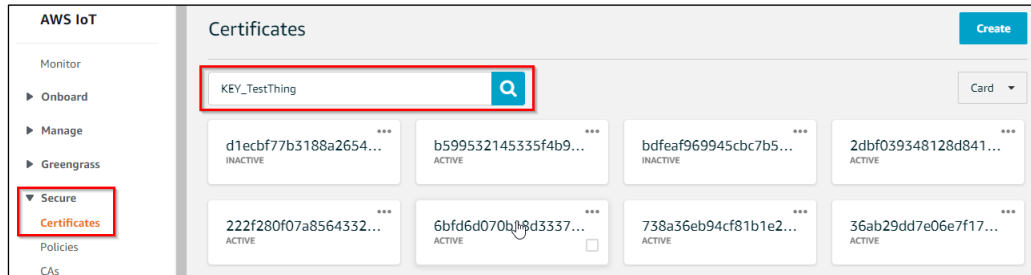
12. Once you click on your certificate, select **Actions > Attach Policy**.

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.
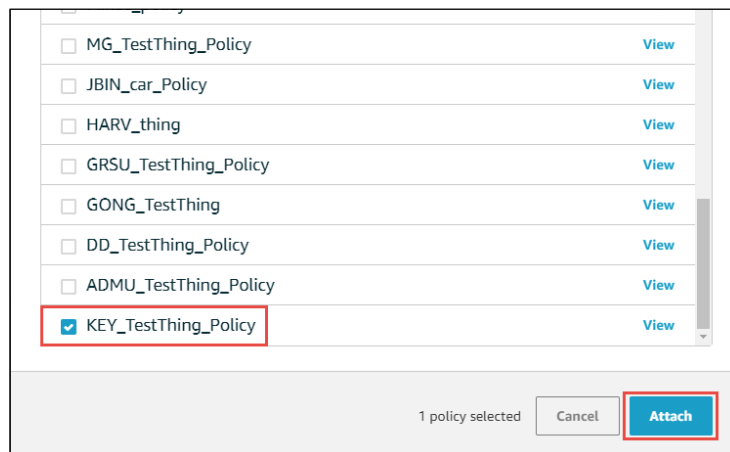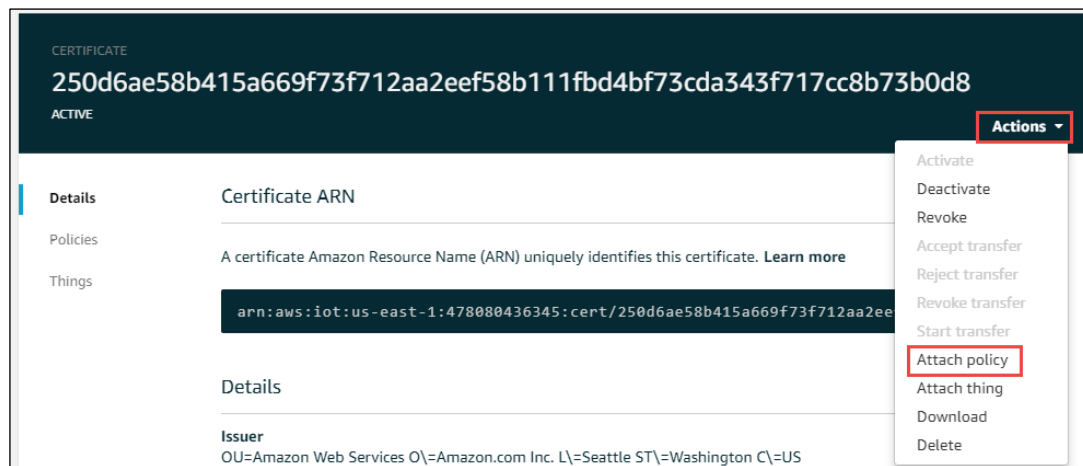
13. Once you get to this point, you should verify:

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 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 > MQTT test client** from the panel on the left side of the screen.

2. Enter a topic that you want to subscribe to such as **<your_initials>_testtopic** in the **Subscription topic** box.

3. Select **Display payloads as strings**, and click on **Subscribe to topic**.
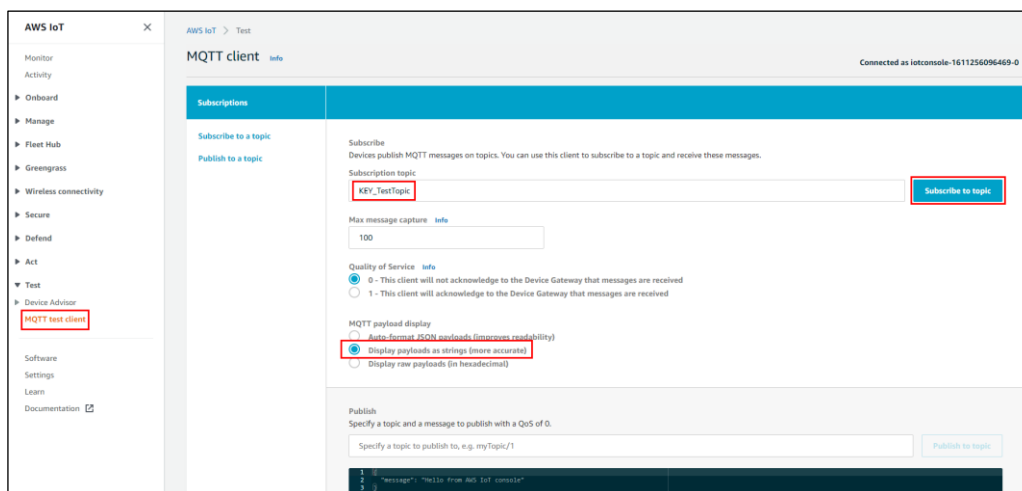

Publishing to a Topic from the Test client

Now that you are subscribed to a topic you can publish messages to that topic from another instance of the MQTT test client.

1. First, open another web browser tab, login to the AWS account, and go to the Test page.

2. 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 (topic names are case sensitive).

3. Then type in your message and press **Publish to topic**. You can see in the box below I sent a JSON message containing "message":"test message".
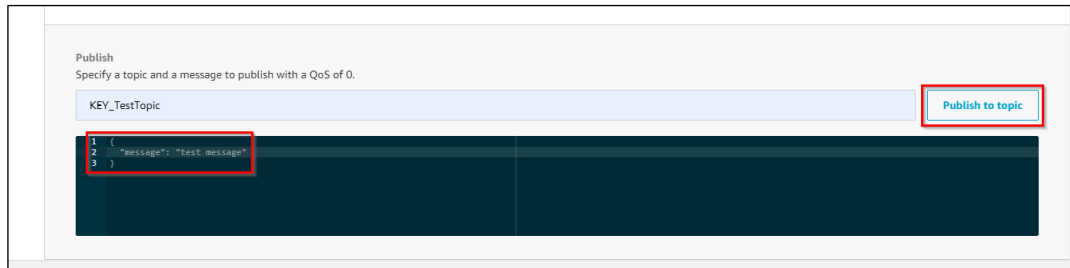
4. Go back to the tab with the subscription and see that the published message was send to the subscriber.
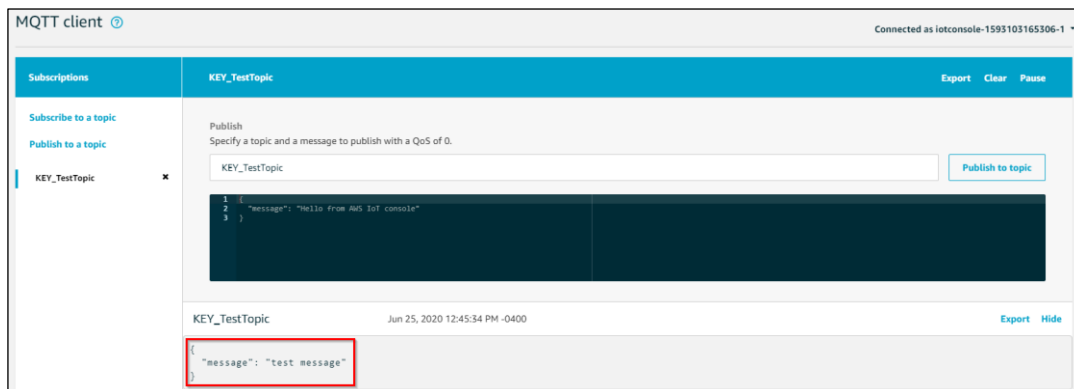
Exercise 4: Run the ModusToolbox™ any cloud MQTT client app

1. Create a New Application within the ModusToolbox™ IDE and select the "AnyCloud MQTT Client" example application. This application has a task to publish and a separate task to subscribe.

2. Modify the *wifi_config.h* file for your network.

   Note: wifi_config.h is in the configs directory.

3. Copy the device certificate, private key, and Root CA certificate that you generated and saved previously into the proper locations in *mqtt_client_config.h.*

They must be formatted as shown below:

```
179    /* PEM-encoded Root CA certificate */
180⊖  #define ROOT_CA_CERTIFICATE        \
181    "-----BEGIN CERTIFICATE-----\n" \
182    "MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n" \
183    "ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6\n" \
184    "b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL\n" \
185    "MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv\n" \
186    "b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj\n" \
187    "ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM\n" \
188    "9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw\n" \
189    "IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6\n" \
190    "VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L\n" \
191    "93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm\n" \
192    "jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC\n" \
193    "AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA\n" \
194    "A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI\n" \
195    "U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs\n" \
196    "N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv\n" \
197    "o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU\n" \
198    "5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy\n" \
199    "rqXRfboQnoZsG4q5WTP468SQvvG5\n" \
200    "-----END CERTIFICATE-----"
201
```

You can manually format the strings as shown above, or you can use one of the following two methods:

I. Use the Python script called *format_certificates.py* provided in the *Scripts* directory with the class material. To use it:

     a) Put the script and certificates/keys in the same directory.

     b) The private key file name must end with *private.pem.key*, the certificate file name must end with *certificate.pem.crt* and the Amazon Root CA must end with *CA1.pem.*

     c) Open modus-shell, go to the directory with the script and enter:

     d) `python ./format_certificates.py`

     e) Copy/paste the formatted strings from the output window to the proper locations in *mqtt_client_config.h*.

     f) Type Ctrl-C in the modus-shell window when done.

II. If you have the Amazon FreeRTOS repo cloned onto your computer, there is an HTML utility that you can use. This repository is located at: https://github.com/infineon/amazon-freertos. Once you have the repo, the tool is located at: *tools/certificate_configuration/PEMfileToCString.html.* To use it:

     a) Open the HTML file in a web browser.

     b) Load each certificate/key file one at a time.

     c) Copy/paste the formatted strings from the output window to the proper locations in *mqtt_client_config.h*.

Note: *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. Set the `#define` for `MQTT_BROKER_ADDRESS` to your broker address in *mqtt_client_config.h*.

The correct address can be found by clicking on **Settings** at the lower left corner of the Amazon AWS IoT Core console window. The broker address is listed as the **Endpoint**.

```
49    /* MQTT Broker/Server address and port used for the MQTT connection. */
50    #define MQTT_BROKER_ADDRESS                "amk6m51qrxr2u-ats.iot.us-east-1.amazonaws.com"
```

5. Modify the `#define` for `MQTT_TOPIC`.

This is the topic that your device will publish and subscribe to. You can use the topic from the previous exercise with your initials in the name.

☐   6.   Modify `MQTT_CLIENT_IDENTIFIER` to the name of the *thing* that you created previously.

> *Note:*   *This a good practice to prevent conflicts between multiple devices on a broker – every device connected to a broker MUST have a client identifier. However, there is a function defined in mqtt_task.c that will append a timestamp onto your client identifier prefix in order to prevent multiple clients with identical identifiers.*

☐   7.   Build and program your project.

☐   8.   Open the serial port and watch your terminal session.

☐   9.   Subscribe to your topic using the AWS MQTT Test client.

    When you press the button on your device, you should see updates to the topic in the test window.

Exercise 5: ModusToolbox™ any cloud AWS MQTT firmware flow

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

☐   1.   How do the AWS library functions (e.g. `IotMqtt_PublishSync`) get into your project?

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

☐   3.   How does the button callback unlock the publisher task?

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

☐   5.   What steps are required to get an AWS connection established?

☐   6.   What function is called to send data to the server?

Exercise 6: Publish from AWS test MQTT client to toggle kit LED

Publish messages using the AWS test MQTT client to test the subscribe task in the application.

☐   1.   Determine what string needs to be sent to turn the light on or off.

Note: *Look in the source code to find the string that is being looked for when a message is received.*

Note: *If you are successful, the LED on your device should turn on/off.*

Exercise 7: 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.

Note: *Start with the httpbin.org GET project using TLS from the HTTP chapter.*

Note: *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 kit is authorized to connect to the broker and interact with your thing.*

4c.9   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
AWS Forum	https://forums.aws.amazon.com/forum.jspa?forumID=210

4c.10 Appendix

Answers to the questions asked in the exercises above are provided here.

4c.10.1 Exercise 5 Answers

1. How do the AWS library functions (e.g. `IotMqtt_PublishSync`) get into your project?

   The `#include` for *iot_mqtt.h* in the C file cause the AWS library functions to be included in the project.

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

   `isr_button_press()`

3. How does the button callback unlock the publisher task?

   It notifies the publisher task by calling:
   ```
   xTaskNotifyFromISR(publisher_task_handle, new_device_state,
   SetValueWithoutOverwrite, &xHigherPriorityTaskWoken);
   ```

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

   No, but messages that affect the shadow have to be JSON.

5. What steps are required to get an AWS connection established?

   Connect to the network
   Initialize the Cypress Secure Sockets and MQTT libraries
   Set up network info and connection info
   Open the connection using `IotMqtt_Connect()`

6. What function is called to send data to the server?

   `IotMqtt_PublishSync()`

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.