

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

Objective

At this end of Chapter 7b you will understand:

- How Message Queue Telemetry Transport (MQTT) works:
 - How MQTT fits into the TCP/IP Networking Stack
 - The purpose of a **Topic**
 - The role of the **Message Broker**
 - The role of a **Publisher**
 - The role of a **Subscriber**
- How to use the JavaScript Object Notation (JSON) language.
- How the Amazon [AWS MQTT Cloud works](#) including:
 - How to provision “things” (which for semantic reasons, will be notated *thing*, a.k.a. your IoT device, in this chapter) in the Amazon AWS IoT Cloud by creating a *thing*, policies and certificates.
 - AWS MQTT and Security
 - Understand the *thing* shadow
 - How to use the AWS IoT test MQTT Client to subscribe and publish to topics
 - Understand the scope of systems that can be implemented in the AWS Cloud (SNS, Database etc.)
- Understand in **DETAIL** how to write WICED firmware to interact with the AWS IOT Cloud

Fundamentals

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):

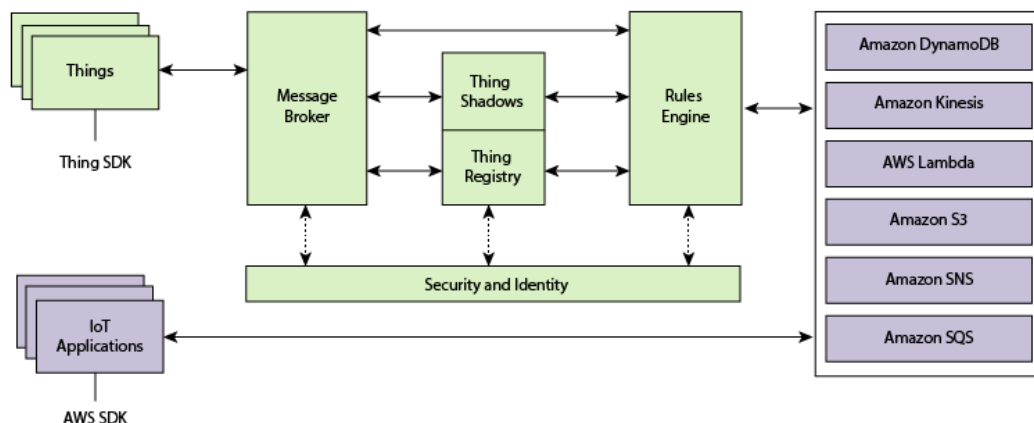
- [AWS IoT](#): A cloud platform that provides Cloud services for IoT devices (the subject of this chapter).
- Amazon Elastic Cloud ([EC2](#)/[EC3](#)): A virtualized compute capability, basically Linux, Windows etc. servers that you can rent.
- [Amazon Lambda](#): A Cloud service that enables you to 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.

Amazon AWS IoT Introduction

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

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



Amazon AWS IoT Resources

There are three types of resources in Amazon 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 using MQTT 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 also specify permissions for specific resources such as topics and shadows. Here is an example of a Policy created for a new *thing*.

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

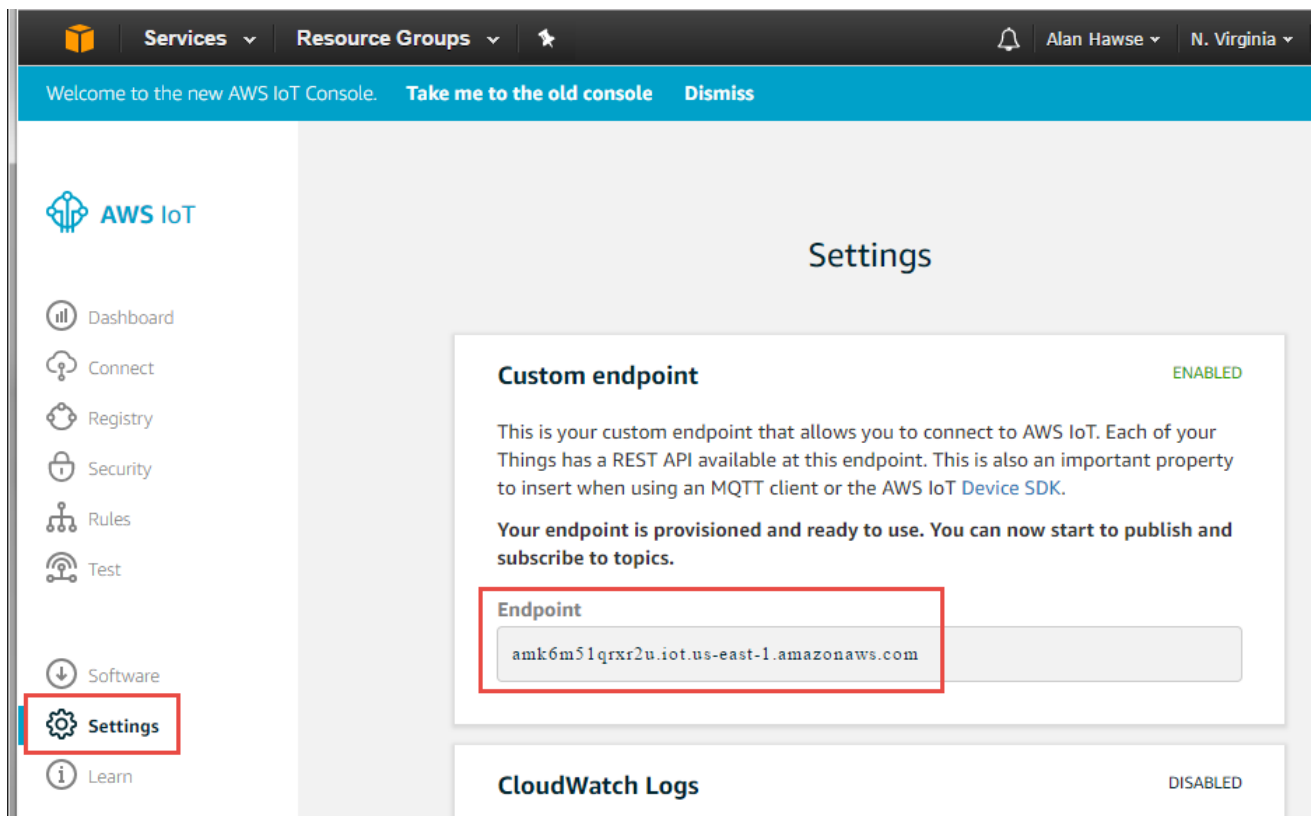
Amazon AWS MQTT

Internet Access

To create a new Amazon 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 an 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. This will go into the firmware as the MQTT_BROKER_ADDRESS.

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.



[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 the 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"}}}
```

[MQTT Topics](#)

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

As the system designer you are responsible for defining what the topics mean and do in your system. Some [best practices](#) 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.

[Device Shadow MQTT Topics](#)

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

MQTT Topic Suffix	Function
/update	The JSON message that you publish to this topic will become the new state of the shadow.
/update/accepted	AWS will publish a message to this topic in response to a message to /update indicating a successful update of the shadow.
/update/documents	When a document is updated via a publish to /update, the complete new document is published to this topic.
/update/rejected	AWS will publish a message to this topic in response to a message to /update indicating a rejected update of the shadow.
/update/delta	After a message is sent to /update, the AWS will send a JSON message 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 using the following topic and message:

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

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

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

If you are using the 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.

Exercise(s)

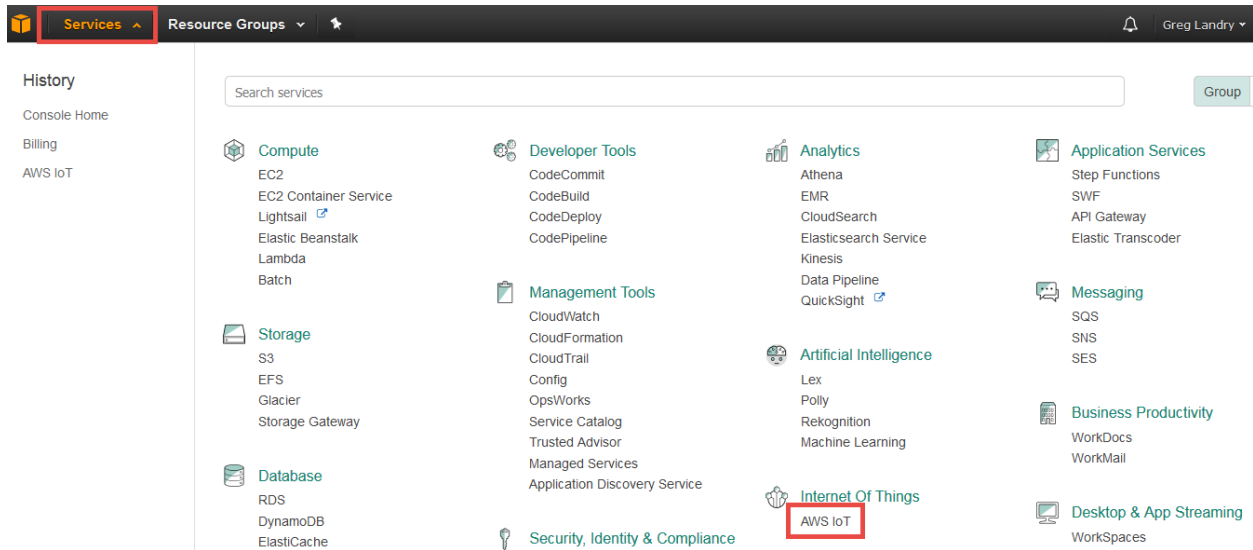
00 Run the MQTT tutorial on the Amazon IoT Console (console.aws.amazon.com)

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

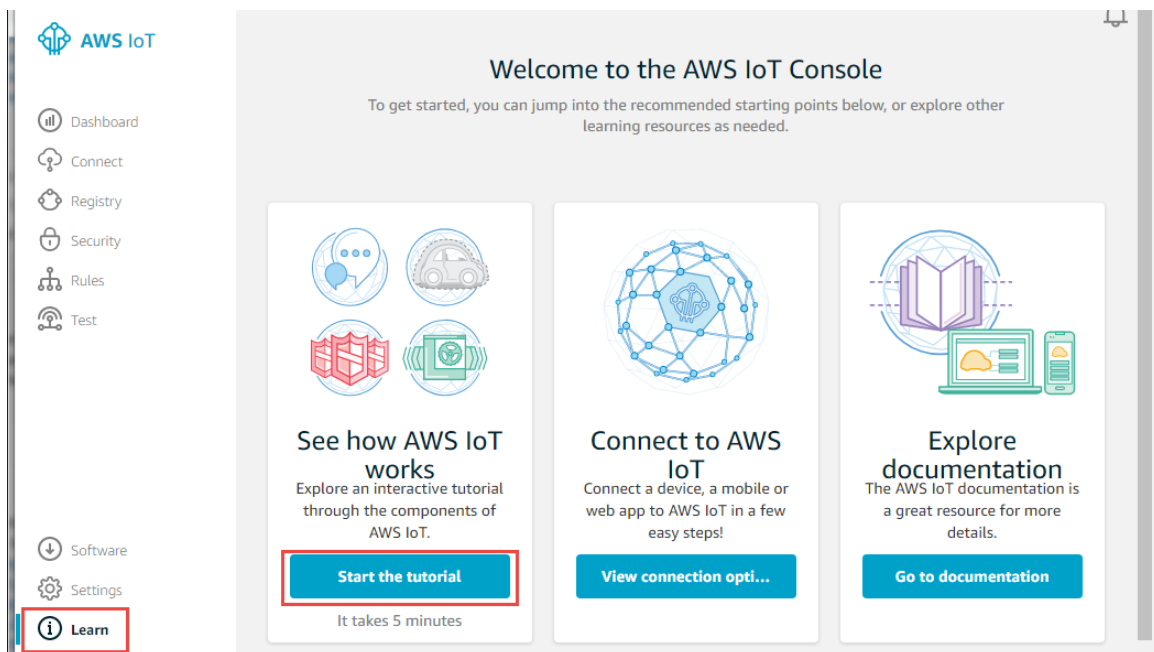
ID: arh@cypress.com

Password: wiced101

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.

The screenshot shows the AWS IoT Interactive Tutorial interface. At the top, there's a navigation bar with 'Services', 'Resource Groups', and a user profile 'Greg Landry' in 'N. Virginia'. The main content area is titled 'HOW IT WORKS AWS IoT Interactive Tutorial'. It features a numbered list of steps: 1. Device Gateway, 2. Rules Engine, 3. Rule actions, 4. Device Shadows, 5. Build solutions, and 6. Done. A diagram illustrates the flow: a light bulb (1) and a control unit (2) are connected to a cloud representing the Device Gateway. The right sidebar contains the heading 'Connect thing to the Device Gateway' and explains that the AWS IoT Device Gateway enables devices to securely and efficiently communicate with AWS IoT. It also mentions that things can communicate via the Device Gateway using different protocols. An example is provided: a connected light bulb and a control unit both connected to the Device Gateway. The control unit can publish commands into the Device Gateway, and the light bulb can subscribe and listen for relevant commands. A 'Try publishing a command from the control unit to see how that command is delivered via the Device Gateway.' prompt is shown. At the bottom, there are 'Exit tutorial' and 'Next' buttons.

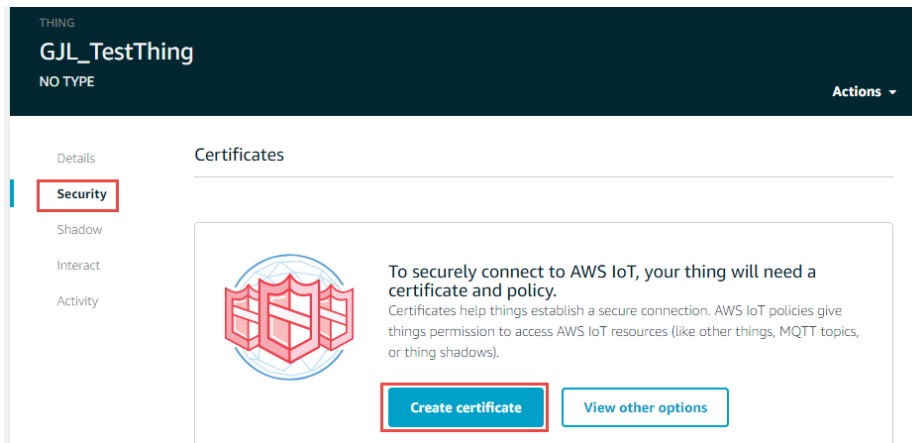
01 Provision a new *thing* in the AWS IOT Cloud, establish its policy and credentials, and test using the AWS test MQTT Client

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 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 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”.

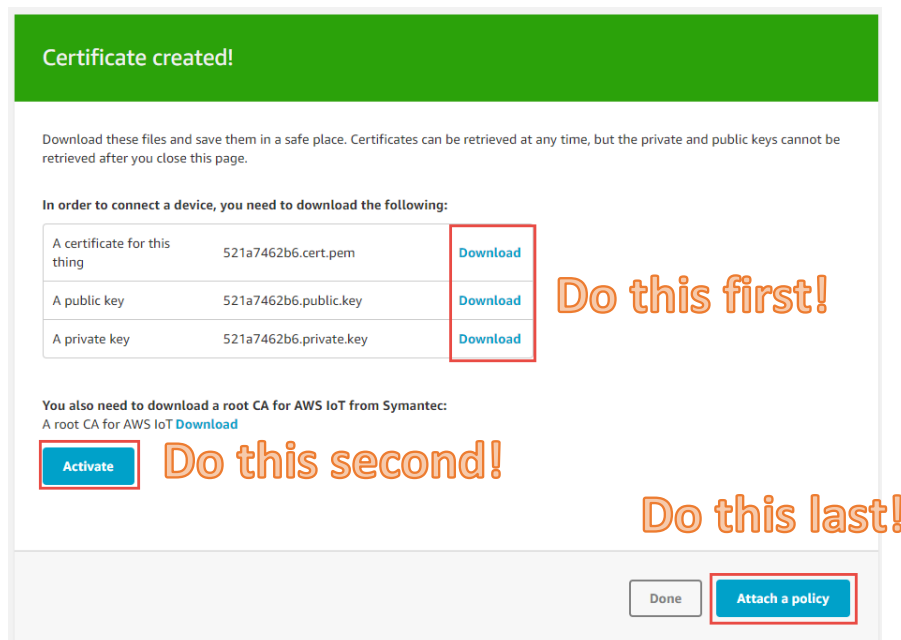
The screenshot shows the 'Register a thing' page in the AWS IoT console. The page has a blue header with a back arrow. Below the header, there's a text box that says 'This step creates an entry in the thing registry and a thing shadow for your device.' The 'Name' field is highlighted with a red box and contains the text 'GJL_TestThing'. Below the name field, there's a 'Show options' dropdown menu. At the bottom right, there's a blue 'Create thing' button, also highlighted with a red box. The top navigation bar shows 'Services', 'Resource Groups', and a user profile 'Alan Hawse' in 'N. Virginia'.

- Before you can access the broker from the 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.



- 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.
- You should also write down the certificate ID 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.



- Click “Attach a policy” and then click on “Create new policy”.
- Give the new policy a name such as “<YourInitials>_TestThing_Policy”. Add the action as “*iot:**”, and 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 (*).

The screenshot shows the AWS IoT Policy console for a policy named **GJL_TestThingPolicy**. On the left sidebar, a red box highlights a back arrow icon. The main content area has a dark header with the policy name and an **Actions** dropdown. Below the header, there's a left sidebar with **Overview** selected, and links for **Certificates** and **Versions**. The **Policy ARN** section shows the ARN: `arn:aws:iot:us-east-1:744830768552:policy/GJL_TestThingPolicy`. The **Policy document** section shows the policy document content, with a red box highlighting the statement:

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

. The **Version 2 updated** section is also visible, along with an **Edit policy document** link.

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.

The screenshot shows the AWS IoT Certificates console. On the left sidebar, the **Security** section is expanded, and **Certificates** is selected, highlighted with a red box. The main content area shows a list of certificates. At the top right, there is a search box labeled *Search certificates* and a **Create** button, both highlighted with red boxes. The list of certificates is as follows:

Certificate ID	Status
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 'CERTIFICATE' and the ARN 'b38392cac7ba3c4429feb51aafcf843804272fb0d78baf65d34d324951013b01'. Below this, the status 'INACTIVE' is shown. A left sidebar contains links for 'Details', 'Policies', and 'Things'. The main content area is titled 'Certificate ARN' and includes a description of the ARN and a 'Learn more' link. Below this, a 'Details' section shows the issuer, subject, and certificate type. An 'Actions' dropdown menu is open on the right, with 'Attach policy' highlighted. Below the main content, a modal dialog titled 'Attach policies to certificate(s)' is displayed. It shows the certificate ARN and a search bar for policies. A table lists available policies, with 'GJL_TestThingPolicy' selected. At the bottom of the dialog, it indicates '1 policy selected' and has 'Cancel' and 'Attach' buttons.

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.

02 Learn how to use the AWS MQTT Test Client

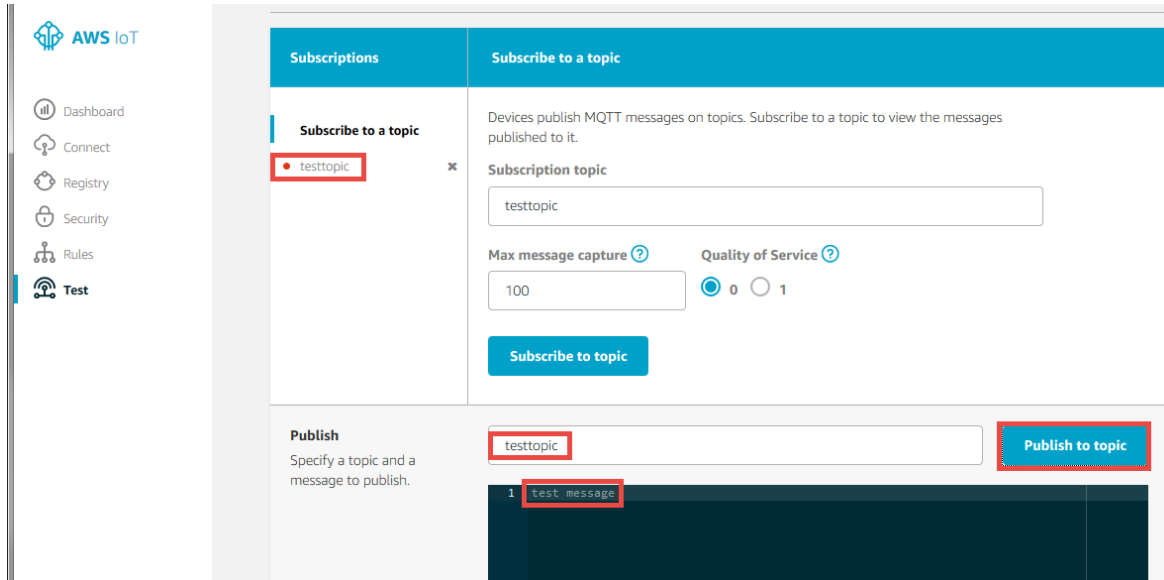
The AWS website has an MQTT Test Client that you can use to test publishing and subscribing to topics. Think of it as a terminal window into your message broker, or as a generic IoT *thing* that can publish and subscribe. You will use this client to test the later exercises.

To use the client:

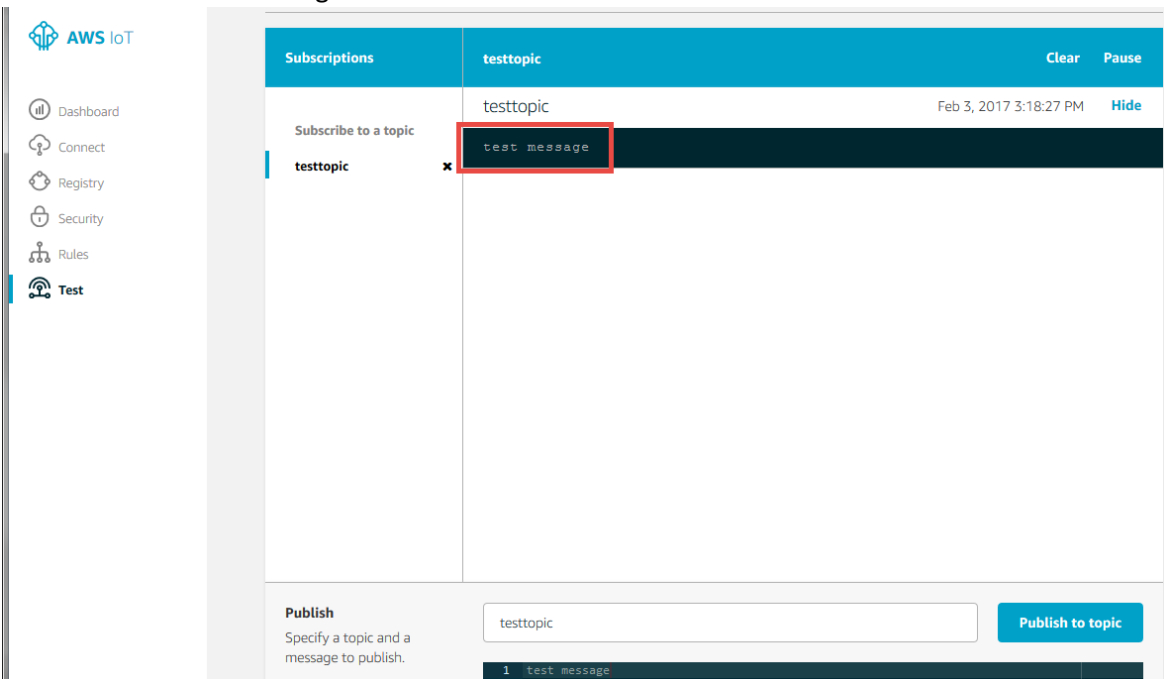
1. Select “Test” from the panel on the left of the screen. Enter a topic that you want to subscribe to such as “<your_initials>_testtopic” and click on “Subscribe to topic”. You will see the new topic show up under Subscriptions. Make sure to put your initials or some other unique string in the topic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

The screenshot displays the AWS IoT MQTT client interface. On the left sidebar, the 'Test' tab is selected. The main panel is titled 'MQTT client' and shows a 'Subscriptions' list with 'testtopic' added. The 'Subscribe to a topic' section is active, showing the 'Subscription topic' as 'testtopic'. The 'Max message capture' is set to 100, and the 'Quality of Service' is set to 0. The 'Subscribe to topic' button is highlighted. Below this, the 'Publish' section shows a message being published to the topic, with the message content: `{ "message": "Hello from AWS IoT console" }`.

- Now that I am subscribed to a topic I can publish messages to that topic from the 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".



- Once the message is sent, you will see a red dot next to the topic in the Subscriptions area. This indicates that you have a new message on that subscription. Click on <your_initials>_testtopic to see the new message.



03 Build and test the demo.aws_iot.pub_sub.publisher App

1. Copy the WICED apps/demo/aws_iot/pub_sub/publisher project to your own directory (i.e. ww101/07b/02_publisher) and update all of the files.
 - a. Hint: Make sure you add your platform to the valid platforms in the makefile and remove all other platforms.
2. Modify the DCT for your network.
3. Change WICED_BUTTON1 to WICED_SH_MB1 to work with the shield button.
4. Copy the certificates that you generated in (01) 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 <i>thing</i> 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.

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.

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.

5. 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.
6. Create a Make Target for your project.
7. Modify the #define for MQTT_BROKER_ADDRESS. This can be found by clicking on "Settings" at the lower left corner of the main console window. The broker address is listed as the "Endpoint".
8. Modify the #define for WICED_TOPIC. Use the topic from (02) with your initials in the name.
9. Modify the #define for CLIENT_ID to include your initials. This is necessary to prevent conflicts since everyone is using the same class broker.
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.

04 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?

05 Build and test the demo.aws_iot.pub_sub.subscriber App

1. Copy the WICED application from apps/demo/aws_iot/pub_sub/subscriber to your directory (i.e. wa101/07b/05_subscriber) and modify the DCT and makefile.
2. Update the topic and broker #defines to the same one you chose for (03).
3. Update the #define for CLIENT_ID to contain your initials.
4. Change WICED_LED1 to WICED_SH_LED1 to work with the shield.
5. We will use the same *thing*, certificate, and keys that we did for (02).
6. 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.

06 (Advanced) Implement the subscriber and publisher in two different kits and test

1. In a real world application, you would typically have one or more devices publishing data to a broker and one or more devices reading data 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. Make a new *thing* in the AWS console for the subscriber and create a new certificate for it. You can attach the same policy that you created previously.
 - i. Note: You could actually use the same *thing*, certificate, and policy for both the subscriber and publisher if you wanted, but in many cases you will want each type of *thing* to have different permissions or settings. For example, you might want the subscriber to be able to read values but not modify them. In that case, the certificate and the attached policy for the subscriber *thing* would be different.
 - ii. Hint: Make sure the CLIENT_ID is different between the two projects. Otherwise they will interfere with one another.
 - b. Save the new subscriber certificate files but use different names for *client.cer* and *privkey.cer* so that the subscriber and publisher can use different files.
 - c. Update the makefile so that the subscriber points to the new certificates.
 - i. Hint: The rootca.cer will not need to change since it is the Amazon AWS public key which is always the same.
 - d. Update the lines in "subscriber.c" that point to the new credential and key files.
 - i. Hint: the credentials are listed as *resources_apps_DIR_aws_iot_DIR_client_cer* and *resources_apps_DIR_aws_iot_DIR_privkey_cer*. These names are the path in the resources folder where folder names are separated by the keyword "_DIR_" and the period before cer is replaced with "_". You could move the credentials to another location in the resources folder by following the naming convention or just change the names of the files and put them in the same folder.
 - e. Program the updated subscriber firmware.
 - f. Power up both kits.
 - g. Subscribe to the topic that you chose using the AWS test MQTT Client.

- h. Press the button on the publisher and watch it change the state of the LED on the subscriber. Also watch the messages in the test MQTT Client window.

07 (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.
3. Update the DCT to have a Config AP for configuration with an SSID name that is unique (so as not to collide with others in your class).
4. Change the following in *aws_common.c* and *shadow.c*:
 - a. WICED_BUTTON1 -> WICED_SH_MB1
 - b. WICED_LED1 -> WICED_SH_LED1
 - c. WICED_LED2 -> WICED_SH_LED0
5. 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.
6. Program the kit.
7. 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.
8. Attach to a Wi-Fi access point from your computer.
9. Go to console.aws.amazon.com and go to the test MQTT Client.
10. Subscribe to the device's shadow topics.
 - a. Hint: *\$aws/things/<YourThingName>/shadow/#* will subscribe to all shadow topics for your *thing*. The # is a wildcard.
11. Press the button on the board and see the messages.
12. Answer the question: What is the sequence of events that changes the LED from On to Off?

References

Resources	Link
AWS Developers Guide	http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html
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

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_shadow	Demonstrates using a shadow device with AWS.

