UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering

EEC 172 **Spring 2019**

LAB 5: Connecting a Thing to the Internet

Verification Due: MW June 5 or TR June 6 at the end of your regular lab section
Report Due, June 6 at 11:00 PM via Canvas upload

Objective:
In this lab, you will explore the RESTful API and use it to connect with Amazon Web Services (AWS). First, you will create a 'thing' in AWS and use the HTTP GET command to retrieve status information about your thing. You will then use your code from Lab 3 to compose messages with an IR remote, which you will send to AWS using the HTTP POST command. You will use a rule in AWS to send the text to your cellphone using Amazon's Simple Notification Service (SNS). Much of the code you will need for accessing AWS will be provided to you. You will need to understand it at a high-level and modify it for your specific AWS account.

Please view the video AWS IoT Getting Started to understand the basics of AWS IoT.

New Equipment Needed
- An Amazon AWS account (You can get a 1 year free trial account, and only 1 is needed per team)

Development Tool Installation Procedure
To do the lab on your own computer, you will need to install the following program:

1. **OpenSSL** (https://www.openssl.org/)
   OpenSSL can be used to convert the .pem formatted certificate and keys to the .der format that is required.

Background on the RESTful API:

The RESTful API, based on the Representational state transfer (REST) architecture, is a web-based communication protocol that is widely used for many services. The RESTful API is typically implemented using Hypertext Transfer Protocol (HTTP). Although not as optimized for low power and low cost applications as another IoT protocol, Message Queuing Telemetry Transport (MQTT), the RESTful API is widely used due to its simplicity and genericity.

The RESTful API defines the format of the messages that are sent to a service to interact with that service. The function of the RESTful APIs varies depending on the service. Some of the commonly used commands are listed below.

Common RESTful API Commands:
- **GET** - Often used to retrieve data or information from a service. May also just be used as a simple trigger. For AWS IoT, GET can be used to retrieve a device's parameters from its shadow.
- **POST** - Can be used to update or push data to the cloud or service. For AWS IoT, POST could be used to update a device's parameters in the device shadow.
- **DELETE** - Used to manage and remove data structures in the cloud. For AWS IoT, DELETE can be used to delete a device shadow.

Other RESTful API commands may exist, depending on the service, and are based off of the HTTP commands: *HEAD, PUT, TRACE, OPTIONS, CONNECT, PATCH.*

For more information on the RESTful API please see: Beginners guide to creating a REST API:
http://www.andrewhavens.com/posts/20/beginners-guide-to-creating-a-rest-api/

Background on AWS IoT:

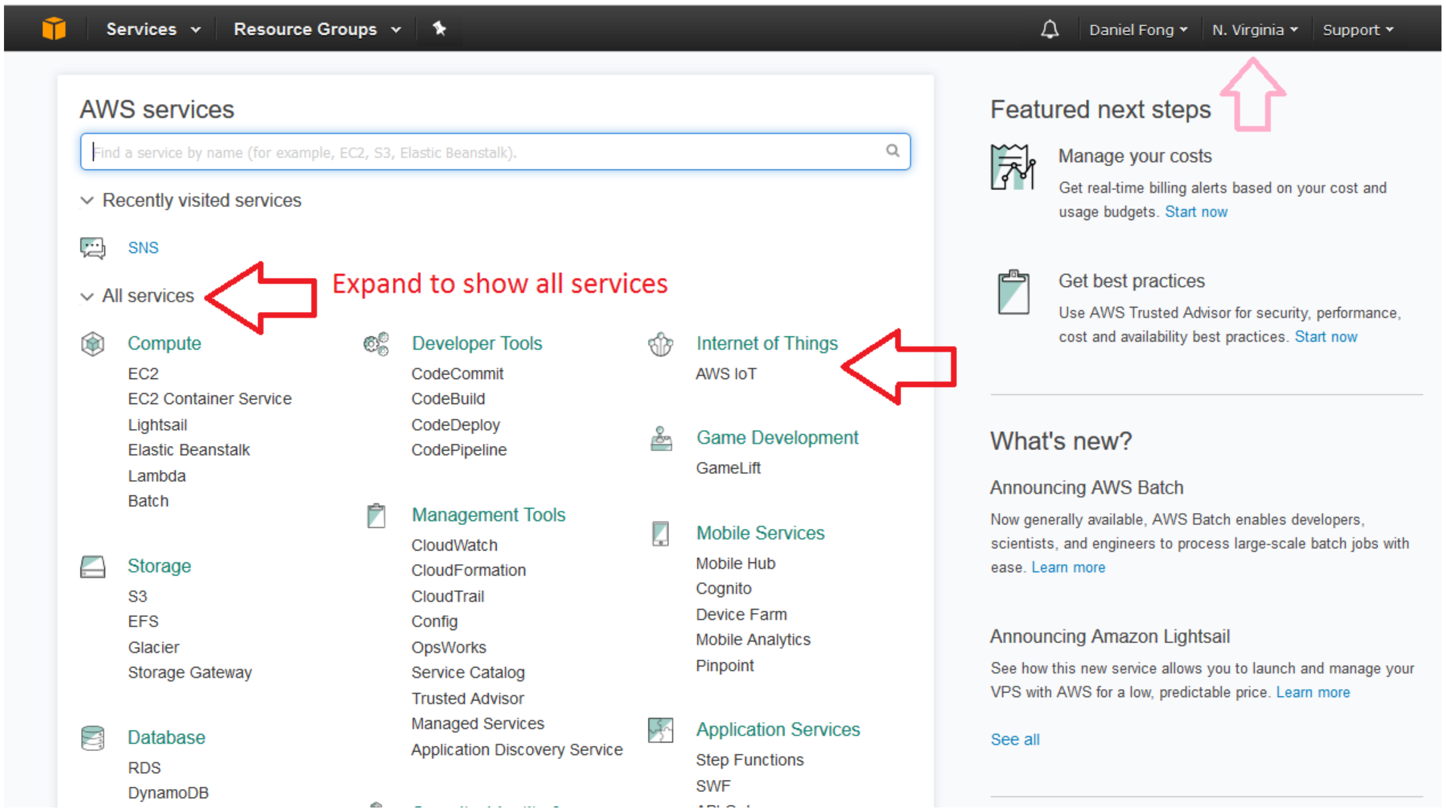
Amazon Web Services (AWS) is a collection of various web-services that removes the need of individual businesses to develop the hardware infrastructure for such services. This includes an Internet of Things (IoT) service, which allows hardware devices to be connected to AWS.

To represent a *real-world* device in the cloud and maintain a persistent/coherent state with intermittent network connections, AWS IoT has a concept of a device *shadow*. The shadow represents the real-world device's state in the cloud. As the device changes states, the real-world device pushes its changes to the shadow device on AWS via the MQTT or the RESTful API. If the device loses network connection, and later comes back online, the device can pull the last-known state from the shadow to update itself. Updates to the shadow can trigger actions in other AWS services as well using *rules*. In this lab, a rule is used to send text messages to your cell phone.

Part I: Securely Connecting to AWS and updating your Shadow

Part I.A: Setting up an Amazon AWS account

For this lab, you will need to setup an Amazon AWS account and get a basic understanding of AWS. One person per team will need to create an AWS account that can be shared for completing this lab. Notify a TA immediately if you or your partner are unable to do so.



Get real-time billing alerts based on your cost and usage budgets. Start now

Use AWS Trusted Advisor for security, performance, cost and availability best practices. Start now

Announcing AWS Batch

Now generally available, AWS Batch enables developers, scientists, and engineers to process large-scale batch jobs with ease. Learn more

Announcing Amazon Lightsail

See how this new service allows you to launch and manage your VPS with AWS for a low, predictable price. Learn more

Figure 1: Amazon AWS Console View

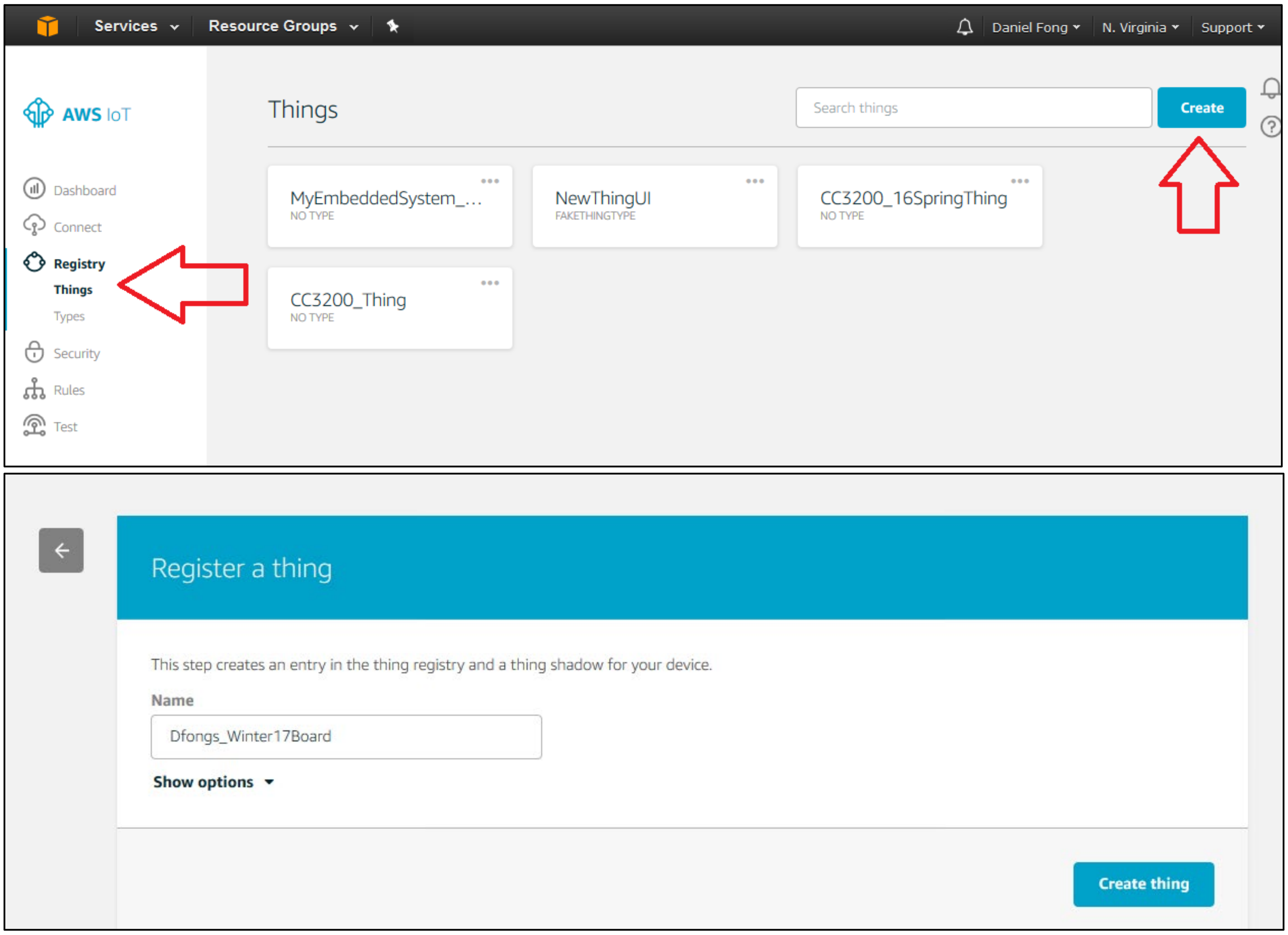
After setting up an account, log into the console. You should end up seeing a page similar to Figure 1. After you are logged in, expand the services and click on the AWS IoT link. If it does not appear, make sure you have selected an Amazon AWS region at which this service is active. We recommend that you use the U.S. East (North Virginia) AWS server for this lab. The controls on the left will list your setup certificates, things, rules, and policies. In the bottom left corner of the screen, there is a *Learn* button and a link to Amazon's IoT *interactive* tutorial. Click on it, and go through the tutorial for a *high-level explanation* about how AWS IoT uses Rules to do cool things.

Part I.B: Setting Up Your First Device Thing/Shadow with the AWS IoT Console

For the next part of the lab, you will need to generate the access keys to allow your CC3200 to connect to your Amazon AWS account securely. For a *real-world* device to connect to Amazon AWS you will need to do the following:

1. **Create a Device Thing/Shadow** (Device name, which can have variables created to interact with the device). This will generate a server-side representation of your *real-world* device (your thing).
2. **Generate the required certificates and keys** for establishing an encrypted connection
3. **Attach the certificate and keys** to the Device Thing/Shadow
4. **Give the Device Thing/Shadow permission** to use the IoT functions of AWS

Fortunately, the above can be accomplished using the Amazon Console interface.



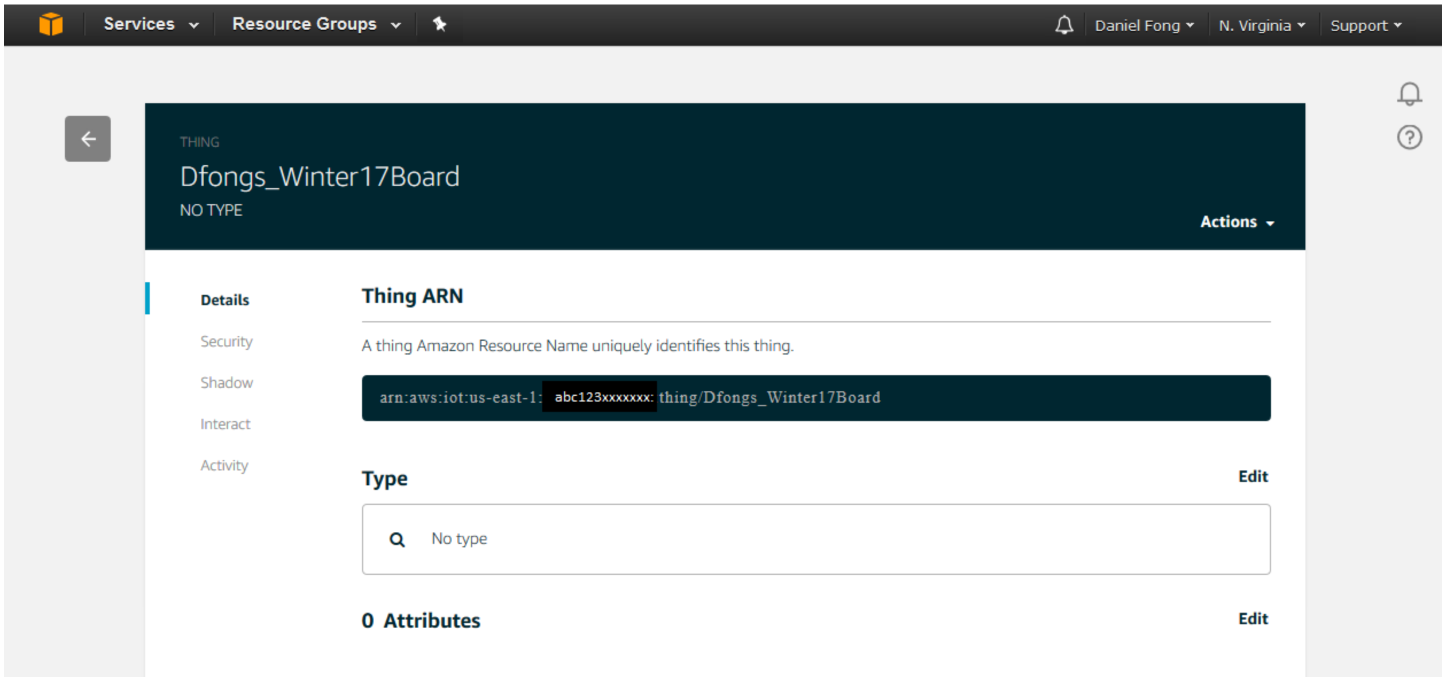
This step creates an entry in the thing registry and a thing shadow for your device.

Name: Dfongs_Winter17Board

Figure 2: Creating a Device Thing

Return to the AWS IoT page and click on *Registry* → *Things* and create a thing. You should be presented with a display similar to Figure 3.

For your first device thing, it is recommended you give it a descriptive name, like 'CC3200_Thing' or '*MyIdName*_CC3200Board' and leave the Type and Attributes options blank. Once you have entered the device name, click create. When successful you should see something that looks like Figure 4.



THING

Dfongs_Winter17Board

NO TYPE

Thing ARN

A thing Amazon Resource Name uniquely identifies this thing.

```
arn:aws:iot:us-east-1: abc123xxxxxxx: thing/Dfongs_Winter17Board
```

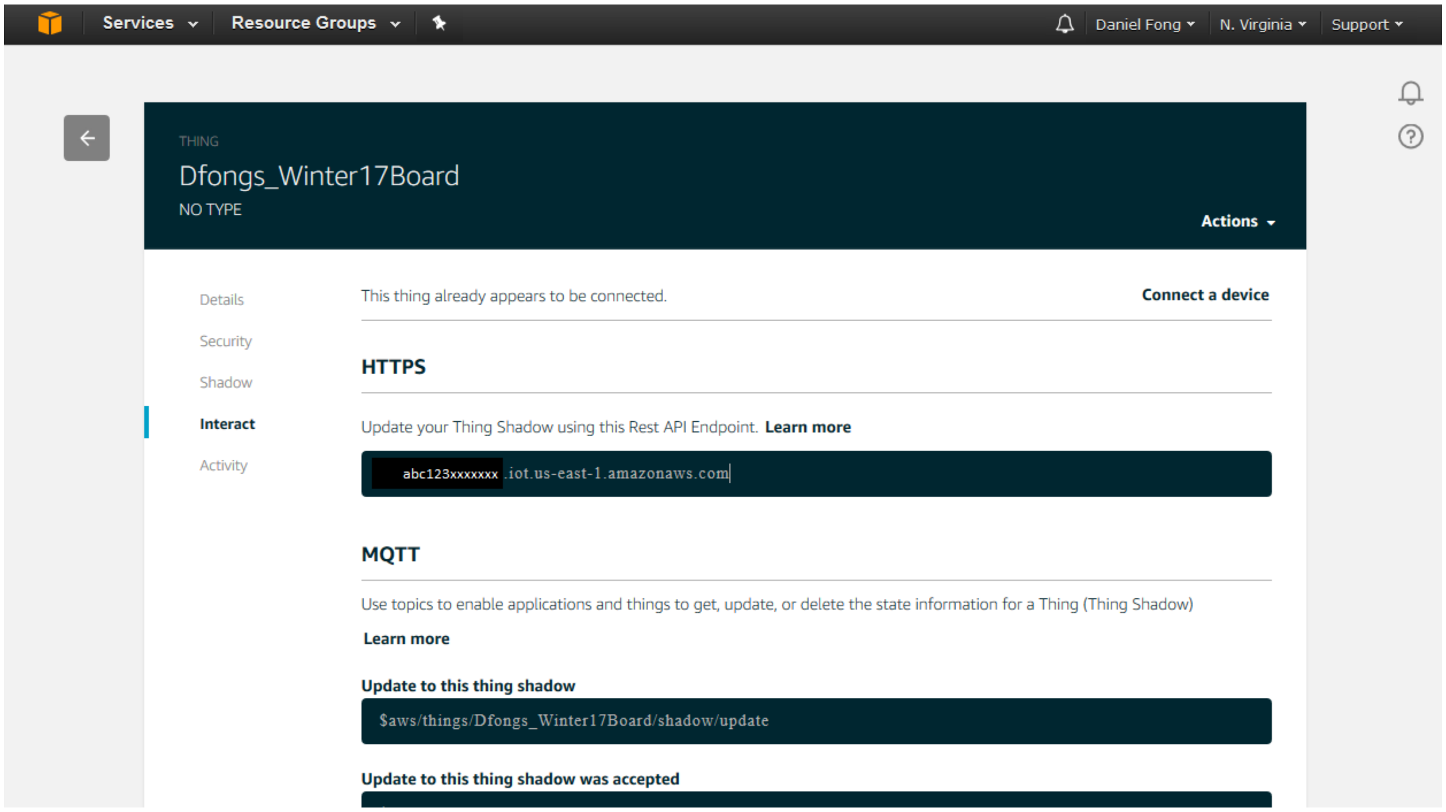
Type

No type

0 Attributes

Figure 3: Successful creation of the CC3200_Thing

Open the *Interact* section on the left. This will display information about how to interact with the AWS representation of the device thing, namely the device *shadow*. You should end up with a view similar to Figure 5.



THING

Dfongs_Winter17Board

NO TYPE

This thing already appears to be connected.

HTTPS

Update your Thing Shadow using this Rest API Endpoint.

```
abc123xxxxxxx .iot.us-east-1.amazonaws.com
```

MQTT

Use topics to enable applications and things to get, update, or delete the state information for a Thing (Thing Shadow)

Update to this thing shadow

```
$aws/things/Dfongs_Winter17Board/shadow/update
```

Update to this thing shadow was accepted

Figure 4: Detailed view of AWS Thing

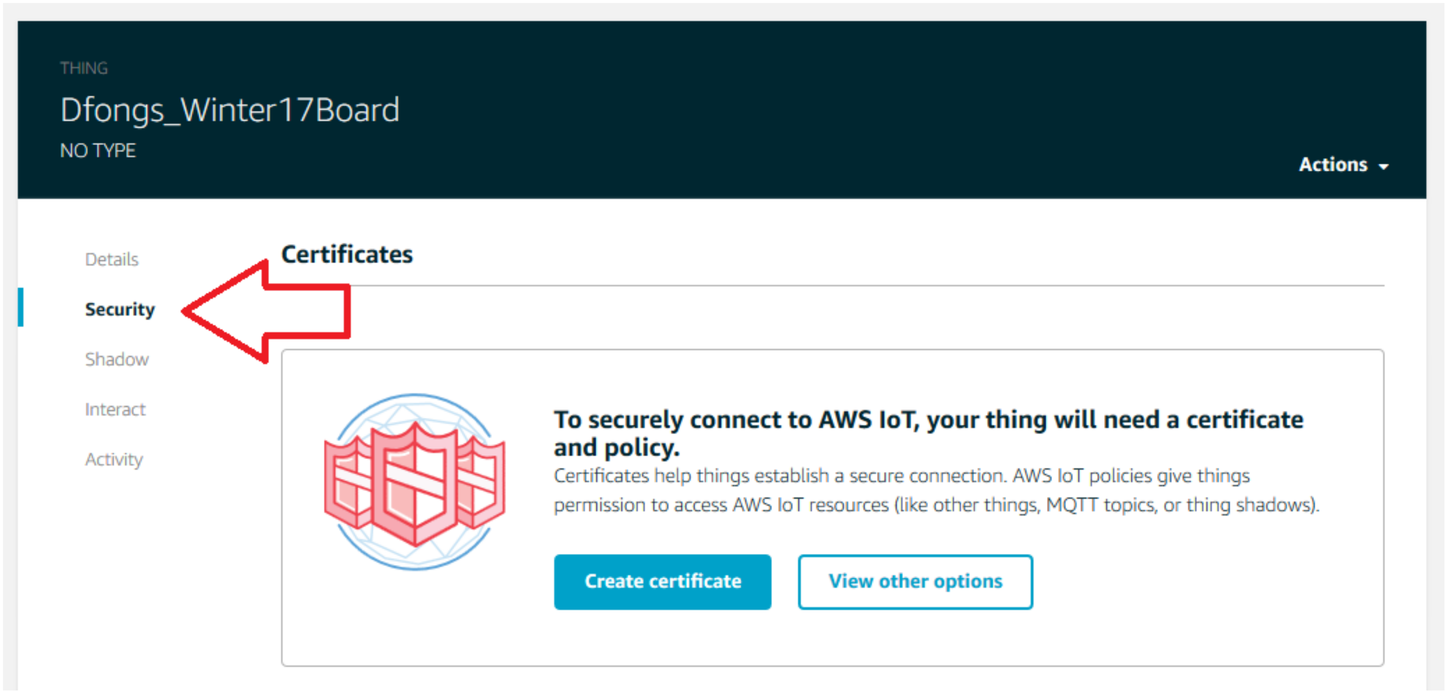
The REST API endpoint displays the web address for the AWS Thing/Shadow, and contains the web address of your account’s AWS Endpoint. For now, copy the endpoint address into a word or text file, and save it for reference. You will need it later to connect to the AWS server. Below is an example Restful API endpoint address with the standard endpoint address for the account highlighted. The initial string of characters will be some randomly generated string generated for your account by Amazon.

Example:
https://xxxxxxxxxxx.iot.us-west-2.amazonaws.com/things/CC3200_Thing/shadow

Updating State Information to your Device Shadow:
`https://endpoint/things/thingName/shadow`

Endpoint:
`identifier.iot.region.amazonaws.com`

After you have recorded your endpoint address, we can **generate the required certificates and keys**, and IoT permissions by opening the *Security*→*Certificates* tab and *Create a certificate*. Use the 'One-click certificate creation' method.


THING

Dfongs_Winter17Board

NO TYPE

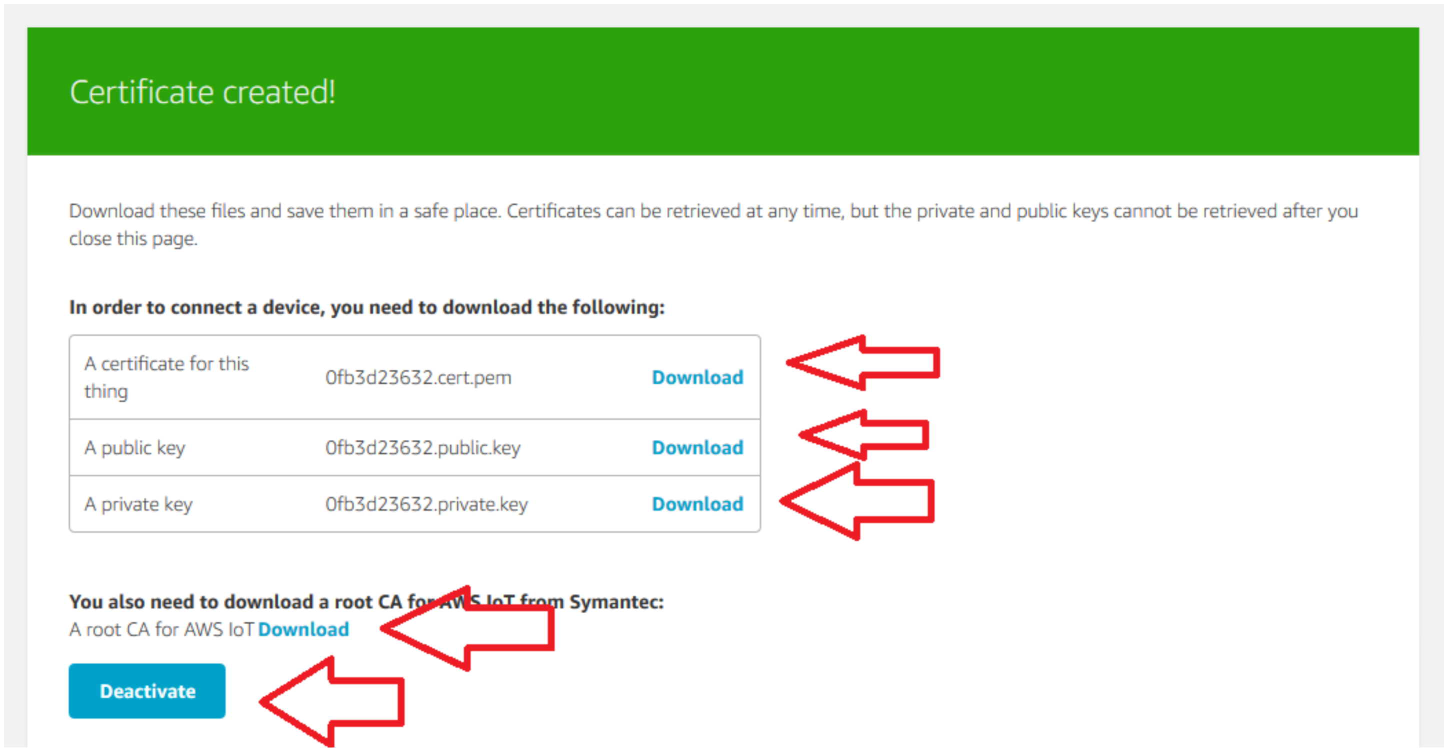
To securely connect to AWS IoT, your thing will need a certificate and policy.

Certificates help things establish a secure connection. AWS IoT policies give things permission to access AWS IoT resources (like other things, MQTT topics, or thing shadows).

Figure 5: Connect a Device Screen

AWS should generate the required public key, private key, and thing certificate for identifying and encrypting traffic from your device, and allow you to download them. **It is important to download and save these files as you will NOT BE ABLE TO RETRIEVE THE PUBLIC and PRIVATE KEYS after you close the page.** The Root CA for Amazon IoT is posted on Canvas: StarfieldClass2CA.crt.der

It is suggested that you save the files in a folder with the name of your Device Thing/Shadow so that it is clear what keys/certificates correspond to which device. After downloading, don't forget to *Activate* the certificate. To learn more about how certificates are used, this explained it very well:
https://realtimelogic.com/blog/2013/10


Certificate created!

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

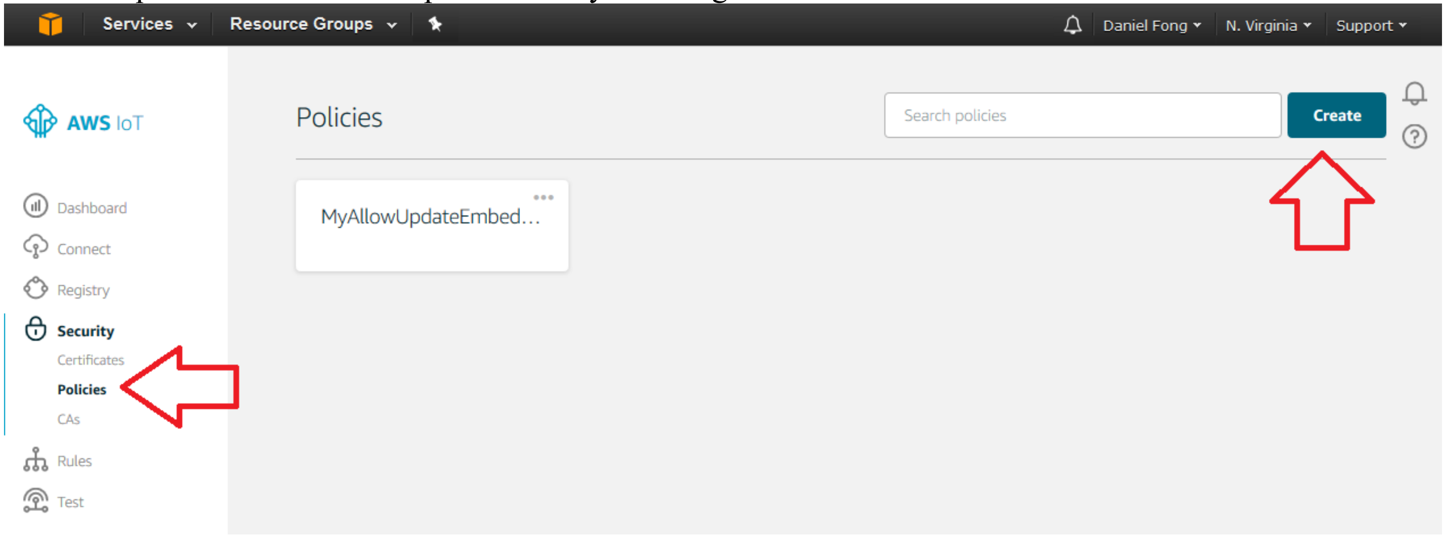
In order to connect a device, you need to download the following:

A certificate for this thing	0fb3d23632.cert.pem	Download
A public key	0fb3d23632.public.key	Download
A private key	0fb3d23632.private.key	Download

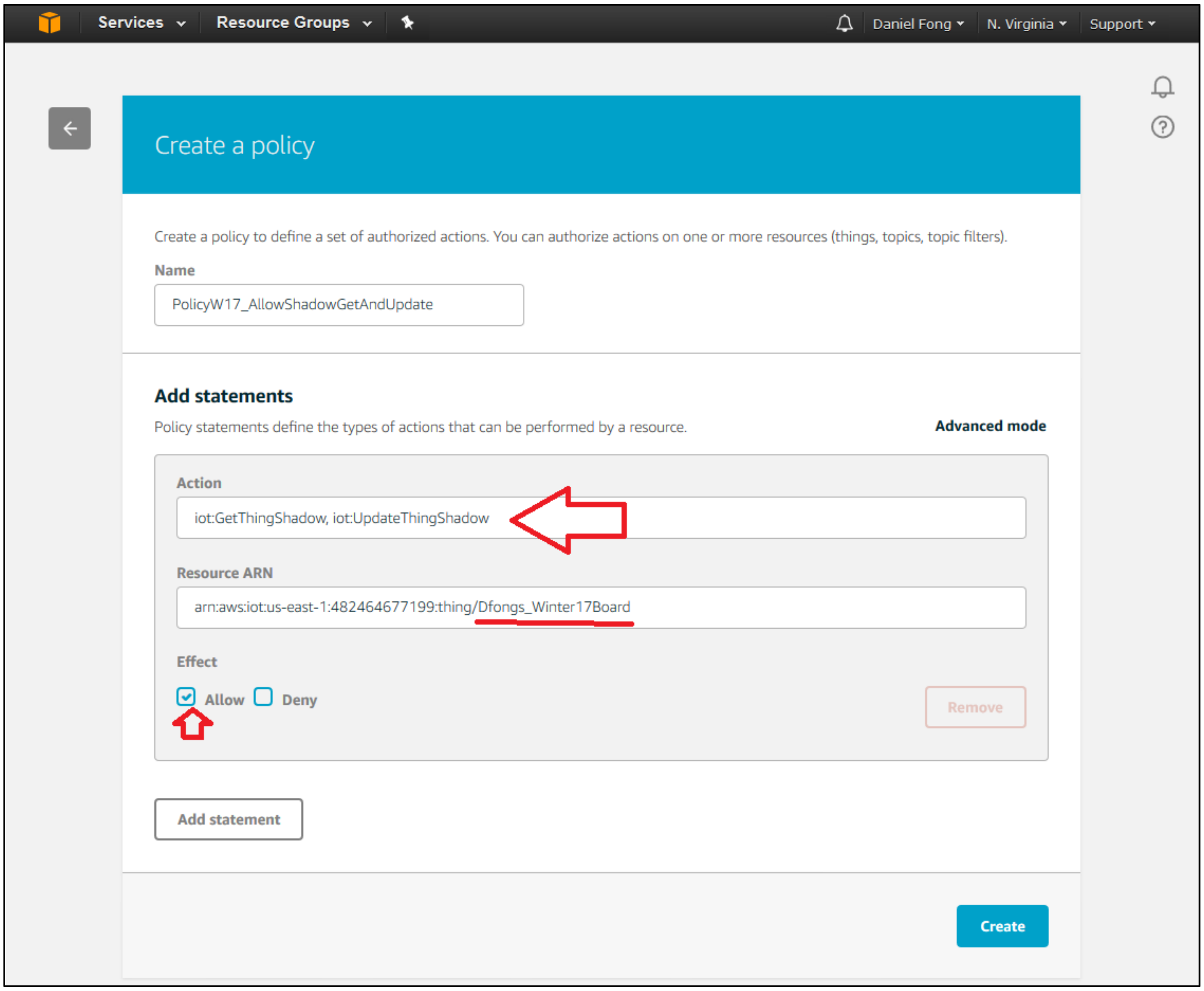
You also need to download a root CA for AWS IoT from Symantec:
A root CA for AWS IoT Download

Figure 6: Download page for generated Certificate and Key files

Part I.C: Making a Policy to Allow Update/Get Status from Thing Shadows
To represent your *thing* (i.e. – our CC3200 board) in the cloud, AWS uses the concept of a *device shadow* to which you can get the last state, or push a new state to the server. Therefore, with intermittent connection to a wireless network, your device could pull the latest state from the cloud and maintain continuity. However, appropriate security measures can be made on AWS to ensure that the devices have the *privilege* to perform the actions requested. These next steps will walk you through how to do this:



On your IoT services panel, navigate to *Security* → *Policies* and create a new policy. Give your policy a descriptive name, and add the actions "iot:GetThingShadow" and "iot:UpdateThingShadow". Furthermore, update the Amazon Resource Number (ARN) to signify that these actions apply to your *thing* created previously. Be sure to select the *Allow* effect and finish creating the policy. You can learn more about the actions and other access management modules in the AWS documentation (http://docs.aws.amazon.com/IAM/latest/UserGuide//access_policies.html).



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: PolicyW17_AllowShadowGetAndUpdate

Add statements

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

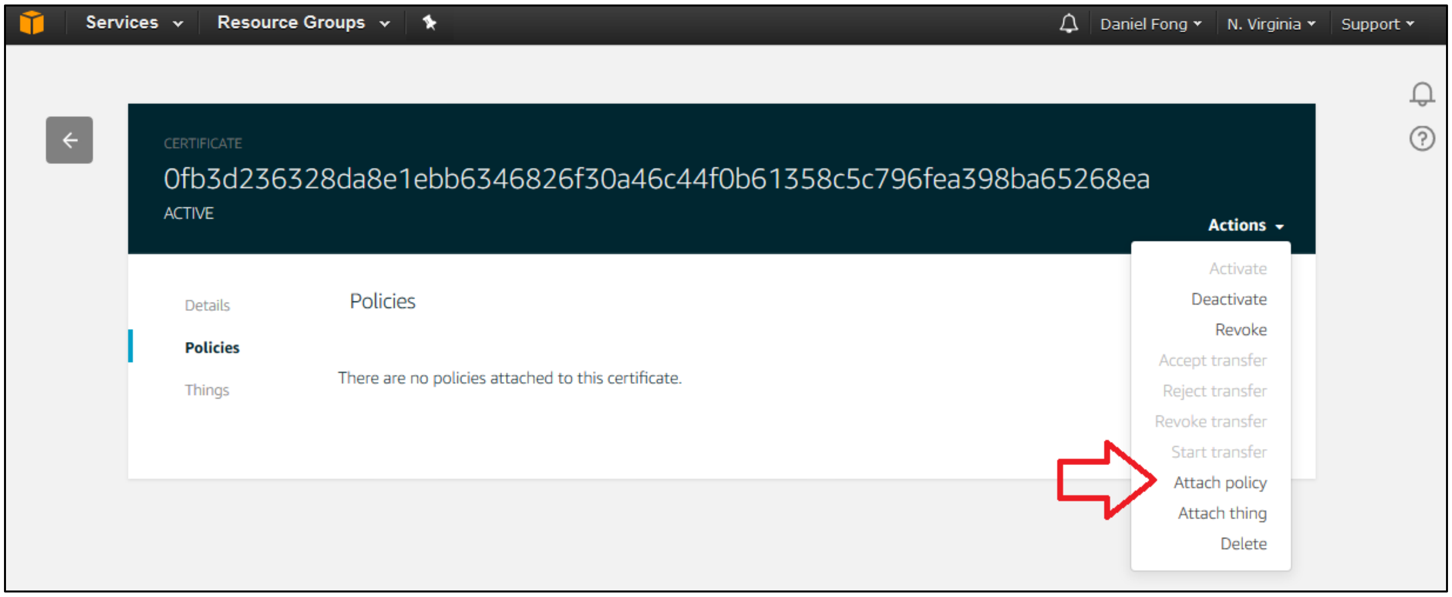
Advanced mode

Action: iot:GetThingShadow, iot:UpdateThingShadow

Resource ARN: arn:aws:iot:us-east-1:482464677199:thing/Dfongs_Winter17Board

Effect: [✓ Allow] Deny

Now that you've created a policy with appropriate privileges for our lab, let's attach it to the certificate your created previously, by navigating to the *Security→Certificates* section and selecting the 'Attach policy' portion (illustrated below). Select the policy you just created and click Attach. This will allow you to use the GET and POST commands via REST from the device associated with this certificate.



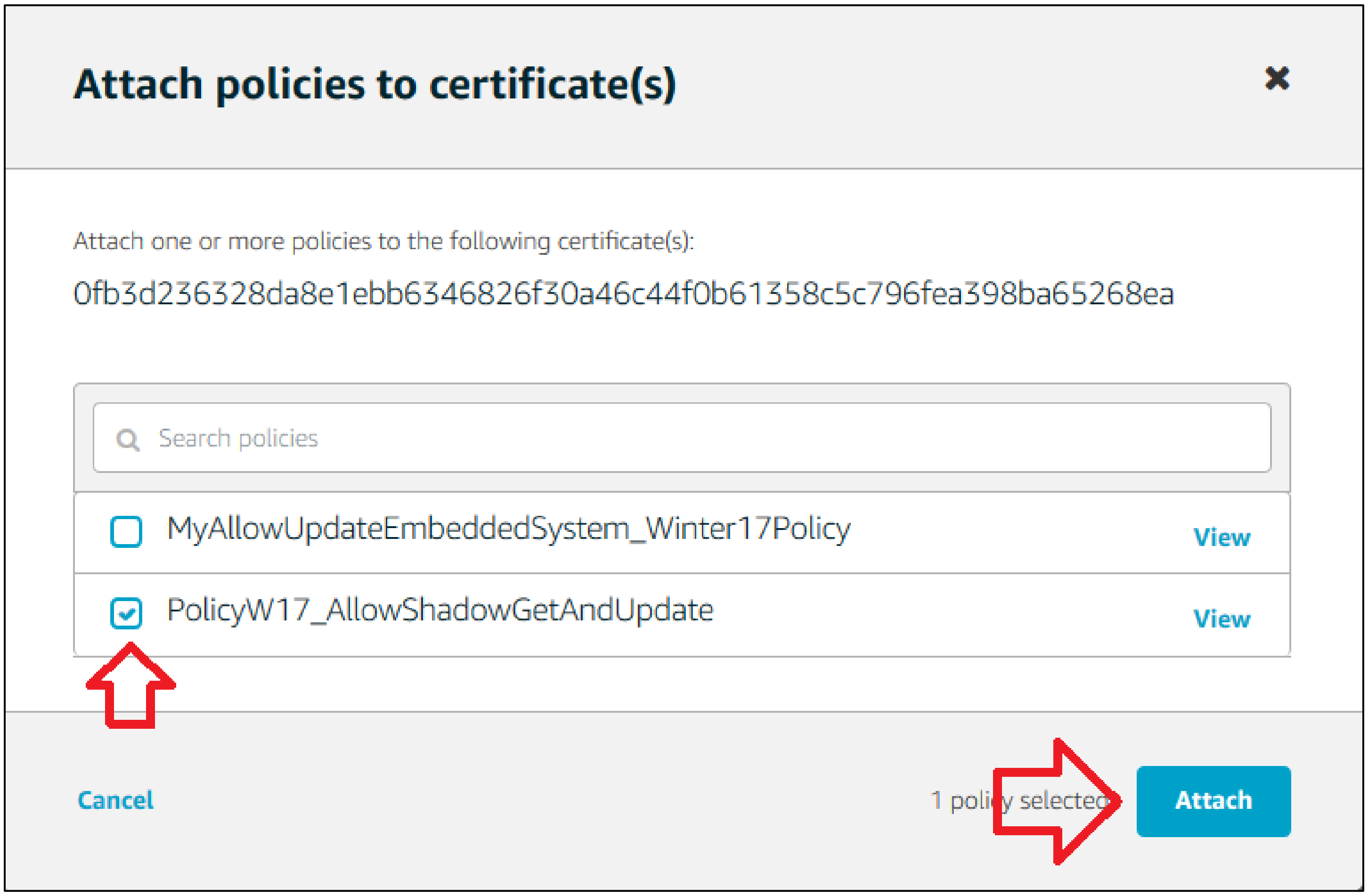
CERTIFICATE

0fb3d236328da8e1ebb6346826f30a46c44f0b61358c5c796fea398ba65268ea

ACTIVE

Policies

There are no policies attached to this certificate.



Attach policies to certificate(s)

Attach one or more policies to the following certificate(s):

0fb3d236328da8e1ebb6346826f30a46c44f0b61358c5c796fea398ba65268ea

Search policies

- [ ] MyAllowUpdateEmbeddedSystem_Winter17Policy — View
- [✓] PolicyW17_AllowShadowGetAndUpdate — View

Cancel

1 policy selected

Attach

Part I.D: Converting the Keys/Certificates for Use with the CC3200
In this part of the lab, you will convert the private key and certificates to another format. This is necessary since AWS uses the .pem format, while your CC3200 uses the .der format.

The connection security between the CC3200 board and AWS is guaranteed through the TLS protocol. There are four components involved, the **public** and **private key**, a **certificate** and a **root certificate**. While the private key is used to decipher encrypted messages, the certificate is used to validate the public key. Certificates are built on a chain of trust, and so there is a root certificate which is used to sign the certificate.

The key and certificates can exist in either .pem or .der format. The former has ASCII-readable characters (unsecure, readable), while the latter is in binary and more compact (still unsecure, but harder to read). Your generated files in AWS are in the .pem format but the CC3200 requires the .der format. The rootCA is already provided for you on Canvas in the Lab 5 folder.

To convert the files from .pem to .der format, we will use openSSL, an open source library for transport layer security. You need to open a command prompt and navigate to the openSSL installation directory. The commands that are needed are the following:

For the certificate file
```
>openssl x509 -outform der -in C:\...\certificate.pem.crt -out C:\...\client.der
```

For the Private Key File:
```
>openssl rsa -outform der -in C:\...\private.pem.key -out C:\...\private.der
```

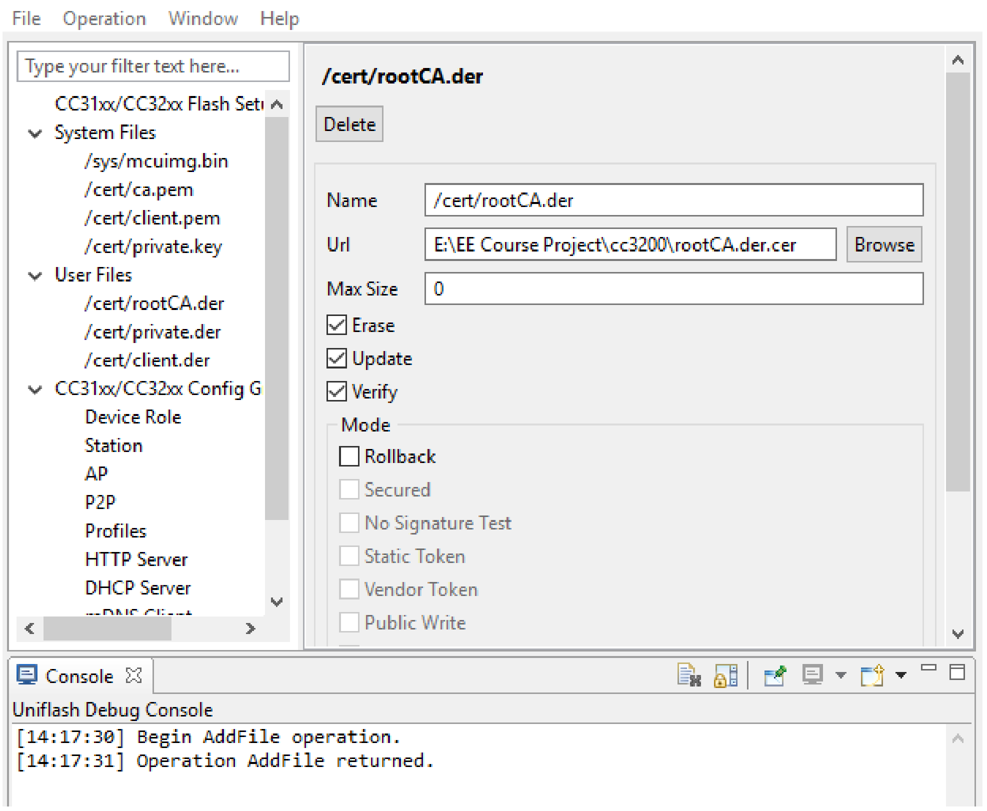
You will obviously need to modify your file names and file paths to where you saved your keys/certificates. If necessary, move the files that need to be converted into the same file directory that openSSL was installed in. The rootCA is shared among all devices connecting to Amazon IoT. The client and private keys should be unique to each device. It is highly recommended to group or name the files accordingly so that the keys/certificate pairs do not get mixed up or over-written.

Part I.E. Using UniFlash to flash key and certificates to CC3200
NOTE: If the firmware on your CC3200 is out of date, you should use UniFlash to update to the latest service pack (2.10.0.0). When you run the application program, you should verify that you get the updated Build Version in one of the status messages, as follows:

```
Host Driver Version: 1.0.1.6
Build Version 2.10.0.0.31.1.4.0.1.1.0.3.34
```

To enable a secure connection from your CC3200, you need to download the private key and certificates to your board. You will do this using the UniFlash utility, as described in the Uniflash tutorial.

You should flash the .der formatted certificates, *root certificate* (used for authenticating the AWS server), *client certificate* (used by AWS to authenticate our device), and the *private key* (for device-side encryption) to **User Files** using the UniFlash utility, as shown in Figure 8. You will specify the path to those files on your PC in the Url box and select the Erase, Update, and Verify checkboxes, and click *Program* to flash those files to the board. You can also load the application (which you will develop in a later part) binary into /sys/mcuimg.bin or you can download it from Code Composer Studio and run it from the Debugger. Take note of the path of your files on the flash (i.e. – the 'Name' field below) as you will need to reference them in your code. If you forget them, you can list the files on your flash by clicking on *List Files* in UniFlash.



Name: /cert/rootCA.der
Url: E:\EE Course Project\cc3200\rootCA.der.cer
Max Size: 0
☑ Erase
☑ Update
☑ Verify

Uniflash Debug Console
```
[14:17:30] Begin AddFile operation.
[14:17:31] Operation AddFile returned.
```

Figure 8: Sample Setup to Flash a Root Certificate Authority File

Part I.F. Accessing AWS using the RESTful API

Amazon's RESTful API for AWS IoT provides access to *device shadows*, or *thing shadows*. These are JavaScript Object Notation (JSON) formatted data that allow state information for AWS things to be stored and retrieved, even when a device might be disconnected from the service. To store or retrieve data from a device shadow, Amazon provides special MQTT feeds and RESTful API calls.

Example GET Method (GetThingShadow)

```
GET /things/thingName/shadow HTTP/1.1
Host: identifier.iot.region.amazonaws.com
Connection: Keep-Alive
```

Example POST Method (UpdateThingShadow)

```
POST /things/thingName/shadow HTTP/1.1
Host: identifier.iot.region.amazonaws.com
Connection: Keep-Alive
Content-Type: application/json; charset=utf-8
Content-Length: 30
{
      "state":{
            "desired":{
                  "color":"green"
            }
      }
}
```

To send a RESTful API POST message to AWS, the service must first be connected to with the SimpleLink socket library. The initial connection should be to the base address for the service or website you are trying to communicate. For example, you may have an AWS device thing that is said to have the RESTful API address of the following: https://<endpoint>.iot.us-east-1.amazonaws.com/things/CC3200_Thing/shadow. The initial connection with the socket library should only be to <endpoint>.iot.us-east-1.amazonaws.com. The HTTPS part of the address will be managed based on the socket credentials, and the AWS device thing subdirectory will be used in the RESTful API calls.

Connection Configuration for AWS

- TLS1.2
- Cipher: TLS ECDHE RSA with AES 256 CBC SHA
- Port: 8443
- Certificates needed: Certificate Authority file for AWS (rootCA.der), x509 Device Certificate file (cert.der) and the matching private key file (private.der)

The RESTful API application requires Internet connectivity through an Access Point (AP). The AP details are contained in the header file common.h. Modify the common.h header file, if necessary, to match the following configuration for the AP in the lab:

```
// Values for below macros shall be modified as per access-point(AP) properties
// SimpleLink device will connect to following AP when application is executed
//
#define SSID_NAME           "eec172"         /* AP SSID */
#define SECURITY_TYPE       SL_SEC_TYPE_OPEN /* Security type (OPEN or WEP or WPA*/
#define SECURITY_KEY        ""               /* Password of the secured AP */
#define SSID_LEN_MAX        32
#define BSSID_LEN_MAX       6
```

Once common.h has been modified, you can build and test the example program. Please note that common.h does not reside in your workspace, so if you work on the lab computers *and* your personal laptop you will need to modify them in both places, and with the appropriate settings for the wireless network you are trying to connect with.

Another important change is that the secure client application must have the "current" date and time so that the credentials can be verified as valid. In this example, we use variables in the main program to update the date and time. The date and time only need to be current within the last month or so. Having the date and time hard-coded into the program is obviously not a robust or secure programming method since the code will eventually stop working unless periodically updated. However, in the interest of reducing complexity, we will use this method for this lab. The date and time variables are stored in the following constants in main.c:

```
//NEED TO UPDATE THIS FOR IT TO WORK!
#define DATE                10   /* Current Date */
#define MONTH               05     /* Month 1-12 */
#define YEAR                2016  /* Current year */
#define HOUR                17    /* Time - hours */
#define MINUTE              0    /* Time - minutes */
#define SECOND              0     /* Time - seconds */
```

Now you can connect your CC3200 securely to AWS. Demonstrate that you can use a POST request to post state to your AWS thing and use a GET request to obtain status information on your AWS thing. You should receive an HTTP 200 status code, showing a successful POST or GET request, as well as some basic status information about your device thing.

NOTE: you need to give your thing shadow some initial state before you can do an HTTP GET on it. You can give the shadow state by using the HTTP POST command or by using the AWS web interface.

In this part, you can start with the working example project SSL_REST_API_AWS.zip. You will need to examine it carefully to understand it and modify it appropriately. In particular, you should examine the http_post() function to see how the HTTP command is sent to AWS. The data must be formatted properly or AWS (or any other web service) will not accept it as valid. In order to use this project on your system, you may need to modify the following files or parameters:

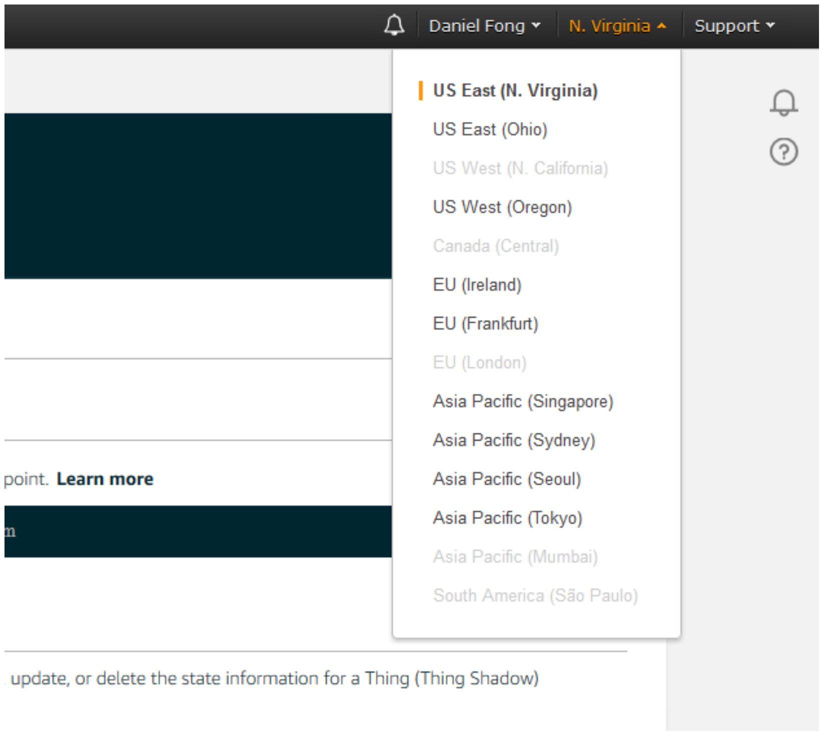
1. common.h – as described above for the eec172 Access Point.
2. main.c – you will need to use your own AWS endpoint path and thing name as well as changing the HTTP command in order to test both GET and POST. You should also change the message data that you post to your thing. However, be careful to keep the correct data format. You may also need to update the date, as described above.
3. pinmux.c – verify that the pinmux.c file sets up the pins that you intend to use. For the default application, only the UART0 and GPIO signals driving the on-board LED are actually used. If your program uses the same pins as the on-board LEDs, you will have problems.
4. Flash the private key and the client certificates and root Certificate Authority into the CC3200 flash, as described.

Demonstrate your working program to your TA and have them sign your verification sheet.

Part II: Using SNS to send a text to your phone

For this part of the project you are to compose a text message using your IR remote and send it to your phone. This will be accomplished by setting up a topic in the Simple Notification Service and registering your text-capable cell phone, and establishing a Rule in the IoT service to trigger when you update your device shadow and send messages to your phone. The interactive tutorial you went through in **Part I.A** gives you a high-level overview on how this is accomplished in AWS.

For this lab we *highly* recommend that you use the "US East (North Virginia)" AWS server. Previously, SMS messages were not enabled on the west coast server yet, and we have not verified it working. The limitation is to select a server that runs both the IoT and SNS service.



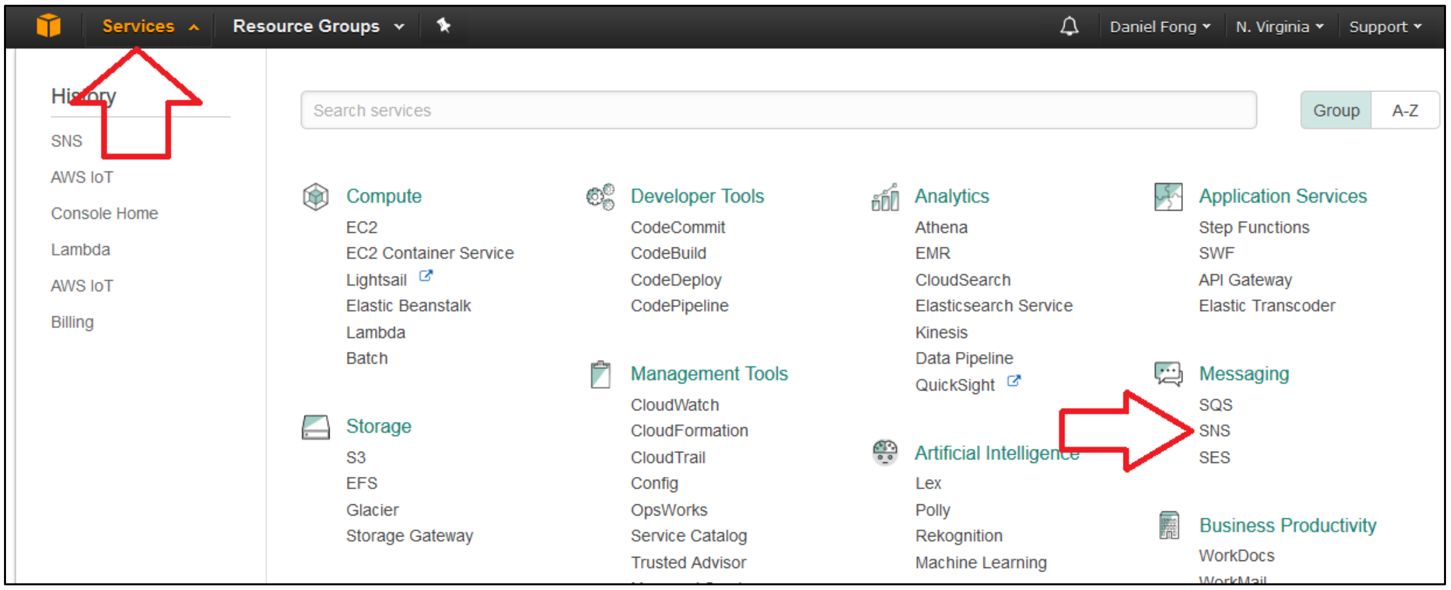
To do this you must perform the following:

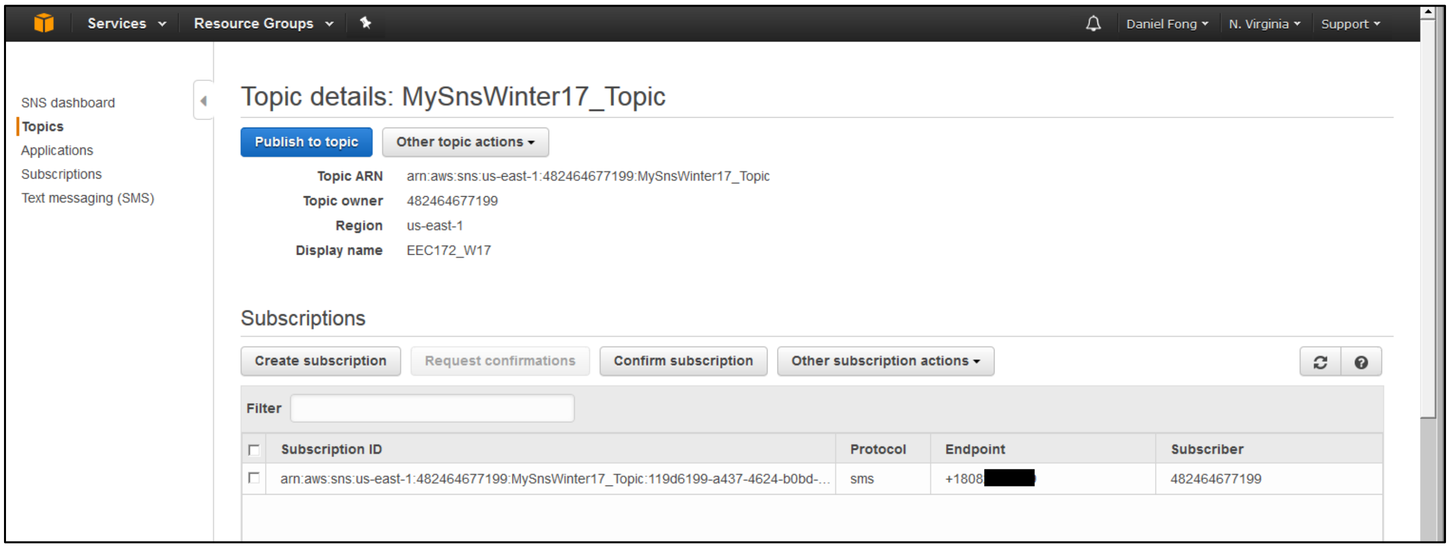
1. Create a thing for the cc3200 as you have done before.
2. Create an SNS topic
3. Subscribe to the SNS topic with your phone number and the SMS option.
4. Create a rule to forward incoming thing messages to your SNS topic.
5. Use the REST API to post to your topic.

We will walk you through **creating an SNS topic** and **creating a rule.**

Part II.A. Creating an SNS Topic:

Amazon Web Services has a collection of different web services which can be integrated to perform a larger function. So far you have been using the IoT service to communicate the states of *real-world devices or things* to the cloud. The Simple Notification Service (SNS) is another module that will allow you to push messages to subscribers. Navigate to the SNS service by expanding the services tab and finding SNS under *Messaging*, and **create a new topic**. Afterwards, **create a subscription**, where the protocol is *SMS* (Simple *Messaging* Service) for text messages. The endpoint is your cell-phone number to receive text messages. You can then **test** to make sure this module works by clicking on the 'Publish to topic' button and manually sending a text message from the AWS Console to your cell-phone (use RAW format for now).





Topic details: MySnsWinter17_Topic

Topic ARN	arn:aws:sns:us-east-1:482464677199:MySnsWinter17_Topic
Topic owner	482464677199
Region	us-east-1
Display name	EEC172_W17

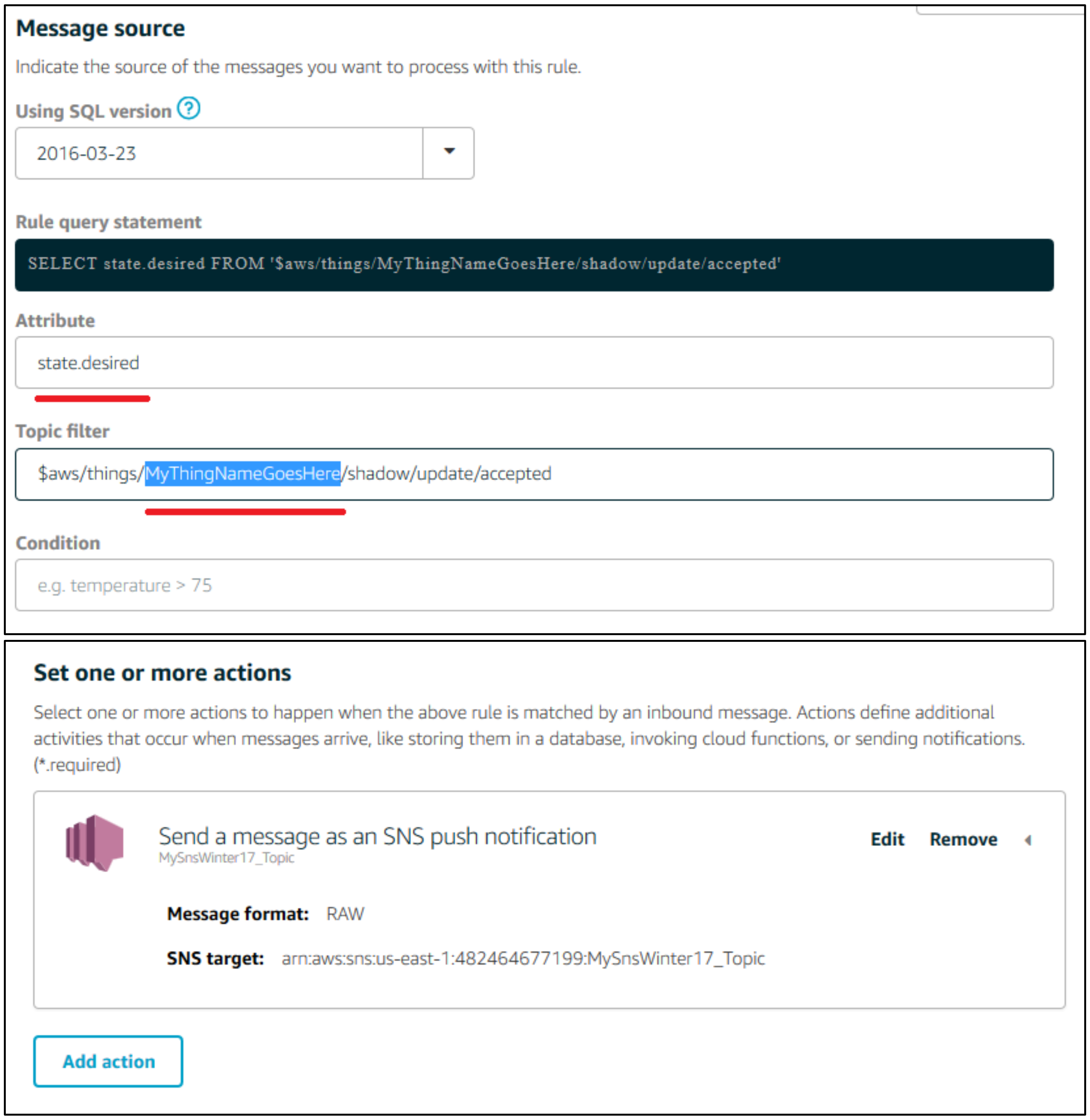
Subscriptions

Subscription ID	Protocol	Endpoint	Subscriber
arn:aws:sns:us-east-1:482464677199:MySnsWinter17_Topic:119d6199-a437-4624-b0bd-...	sms	+1808[redacted]	482464677199

Part II.B. Creating an IoT Rule:

The next step is to create an IoT Rule that will trigger when you push updates to your shadow device. Earlier in the lab, we updated our device shadow (state) via the REST api. When our device shadow is updated, the IoT module publishes messages to several reserved Message Queue Telemetry Transport (MQTT) topics, a publish-subscribe 'lightweight' messaging protocol. In order to trigger your rule, we will be listening for updates to the device shadow via these topics. A full list of reserved topics that can be subscribed to can be found at http://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-mqtt.html .

Navigate to the *Rules* section in the IoT module, and create a new rule. You should listen to the topic '`$aws/things/`*`thingName`*`/shadow/update/accepted`' (the $ should be *included*), report on the attribute '`state.desired`', and leave the condition blank. Then attach an action that will send an SNS push notification when the rule is triggered. Select the SNS target to be the topic you created previously, and use the 'RAW' message format (for now). You will also need to create an IAM role to allow the IoT to access the SNS service 'securely' (you can learn more about Identity Access and Management [IAM] roles here: http://docs.aws.amazon.com/iot/latest/developerguide/iam-users-groups-roles.html, but will not need to understand them for this lab). Your SQL query statement for your rule should be similar to the figure below.


Message source

Indicate the source of the messages you want to process with this rule.

Using SQL version: 2016-03-23

Rule query statement

```
SELECT state.desired FROM '$aws/things/MyThingNameGoesHere/shadow/update/accepted'
```

Attribute: state.desired

Topic filter: $aws/things/MyThingNameGoesHere/shadow/update/accepted

Condition: e.g. temperature > 75

Set one or more actions

Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. (*.required)

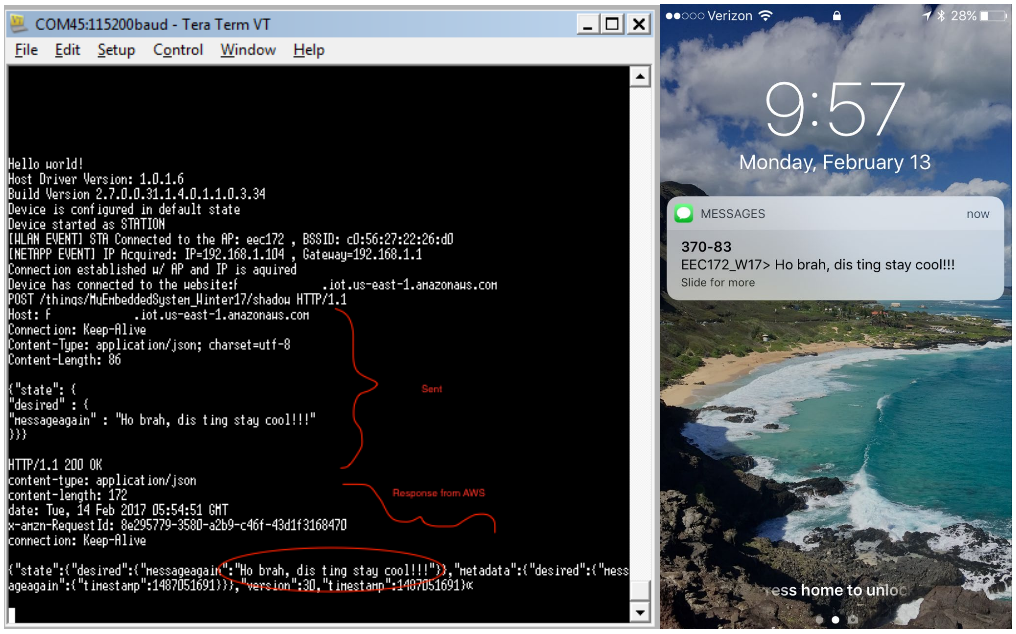
Send a message as an SNS push notification
MySnsWinter17_Topic

Message format: RAW

SNS target: arn:aws:sns:us-east-1:482464677199:MySnsWinter17_Topic

Figure 9. Setting up your IoT Rule

Make sure the rule is enabled and try to push an update to your device shadow using your CC3200,… in a couple of seconds you *should* get a text message. Once that you works, you can clean up the formatting using JSON message format SNS action. To figure out how to do this, go back to the SNS topic you made, click the *Publish a topic* button, and experiment with the JSON message generator to find the appropriate syntax. Investigate how you will need to modify your SQL statement to 'clean-up' your message. Helpful link and subsections for SQL: http://docs.aws.amazon.com/iot/latest/developerguide/iot-sql-reference.html



```
Hello world!
Host Driver Version: 1.0.1.6
Build Version 2.7.0.0.31.1.4.0.1.1.0.3.34
Device is configured in default state
Device started as STATION
[WLAN EVENT] STA Connected to the AP: eec172 , BSSID: c0:56:27:22:26:d0
[NETAPP EVENT] IP Acquired: IP=192.168.1.104 , Gateway=192.168.1.1
Connection established w/ AP and IP is aquired
Device has connected to the website:f             .iot.us-east-1.amazonaws.com
POST /things/MyEmbeddedSystem_Winter17/shadow HTTP/1.1
Host: f             .iot.us-east-1.amazonaws.com
Connection: Keep-Alive
Content-Type: application/json; charset=utf-8
Content-Length: 86

{"state": {
"desired" : {
"messageagain" : "Ho brah, dis ting stay cool!!!"
}}}

HTTP/1.1 200 OK
content-type: application/json
content-length: 172
date: Tue, 14 Feb 2017 05:54:51 GMT
x-amzn-RequestId: 8e295779-3580-a2b9-c46f-43d1f3168470
connection: Keep-Alive

{"state":{"desired":{"messageagain":"Ho brah, dis ting stay cool!!!"}},"metadata":{"desired":{"mess
ageagain":{"timestamp":1487051691}}},"version":30,"timestamp":1487051691}«
```

370-83
EEC172_W17> Ho brah, dis ting stay cool!!!

Part II.C. Integrate your IR Remote multi-tap texting to send a message to your phone:
Now that you know have the SNS module and IoT Rules set up appropriately, you can start integrating your IR remote text messaging code (Lab 3) to send a text to your phone. Start with the program for connecting to AWS via the REST API that you implemented in the last part. When you are done, you will be sending a text from your IR remote, to the CC3200, and then through the cloud to your cellphone. Cool, yeah?

The following links provide useful information on AWS.

Amazon IoT Rules Documentation: http://docs.aws.amazon.com/iot/latest/developerguide/iot-rules.html
Amazon Device Shadows Documentation: http://docs.aws.amazon.com/iot/latest/developerguide/iot-thing-shadows.html

Demonstrate your working program to your TA and have him sign your verification sheet.

Part III: Securely Connecting to Another Web Service

In this part of the lab you are to use the REST API from to connect to a web service other than AWS and do something interesting and creative. You may use any of the equipment from the previous labs, on-board sensors of the CC3200, or other components purchased by yourself.

Requirements:
What you do in this part of the lab is open for you to decide. However, there are some requirements and some goals for this part.

The requirements are:

- Control or monitor something over the internet using the CC3200 LaunchPad. You may use 1 or 2 CC3200 Launchpads in your system.

- You must control or monitor your thing over the internet using some combination of the following devices interfaced to the CC3200:
  - IR remote control for text or numeric input
  - Microphone
  - DTMF input
  - OLED for text or graphics display
  - LEDs
  - Switches
  - Sensors, such as accelerometer, temperature sensor or any external sensors that you interface to the CC3200.

  Your system must be a stand-alone embedded system and must operate without being connected to a host computer. For your checkoff, you must use Uniflash to load your code into the external serial flash instead of downloading from CCS.

The goals are as follows:

- You connect securely to a web-service other than AWS via the internet.

- Another goal is that your system is interactive.

Lab Report

At a minimum your lab report should include:

- Your verification sheet.
- A soft copy of your well-written, *well-commented* final program. Include your names in the header comments of the file containing the main program!
- A soft copy of your code uploaded to SmartSite.
- An outline of your program, with an explanation of how each section of your code works and why it is necessary.
- A description of any noteworthy difficulties you encountered in constructing your solution.

References and Links For AWS Connection in Part 1:

CC32xx SSL Demo Application:
http://processors.wiki.ti.com/index.php/CC32xx_SSL_Demo_Application

Configuring the AWS CLI client to connect to the Amazon Servers:
http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html

Tutorial for using the Identity and Access Management Guide (This is used to generate the credentials to connect via AWS CLI):
http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

IAM tutorial link:
http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

Amazon IoT Quickstart Guide Link:
https://docs.aws.amazon.com/iot/latest/developerguide/iot-quickstart.html
http://docs.aws.amazon.com/iot/latest/developerguide/protocols.html

Resources for Connection to other Services in Part 3:

There are many resources available for your use. For example, you may use example code from the CCS SDK example projects. You must cite whatever resources you use.

An example of an on-line project than might be useful is:

CC3200 HTTP Client Library, OAuth, REST API, TLS Example Code by Glenn Vassallo

https://e2e.ti.com/support/wireless_connectivity/simplelink_wifi_cc31xx_cc32xx/f/968/p/435960/1561493

A useful website for finding the root certificates that are needed for a secure connection to a given webserver is http://ssltools.com. You can download the root certificate either from a link on the webpage (note the ironically that this tool has an insecure connection!) or by going directly to the root Certificate Authority's (CA's) repository.

As an example, checking https://ifttt.com/maker_webhooks using the ssltoots.com identifies the root certificate:

GlobalSign Root CA

The corresponding certificate file can be downloaded directly from GlobalSign's repository. Note that https://ifttt.com/maker_webhooks uses port 443 for secure ssl/tls connections, unlike AWS which uses 8443 (port 443 is the standard port used for SSL/TLS connections). There are many interesting things that can be done using IFTTT.

Note that while you are building your HTTP request to send to the web service, the end of a segment should be ended with ‘\r\n’ (NOT ‘\n\r’). To be clear, the HTTP protocol specifies the end of a line segment to be terminated by a *carriage-return followed by a newline*. The details can be found in the IETF’s RFC2616 (https://tools.ietf.org/html/rfc2616 and https://www.w3.org/Protocols/rfc2616/rfc2616-sec2.html#sec2.2).

You may also use tools like Postman to test your api calls and http request format. Please be aware that copy-pasting values directly into your code sometimes leads to ‘not a character’ errors. To ensure your values are plain-text, please sanitize them using a plain-text editor or typing it in manually (but carefully).

The microphone used in Lab 4 can potentially be connected to a cloud service that does speech to text, such as IBM Speech to Text, and then do something interesting with the resulting text.

There are many interesting web services available to which the CC3200 could be connected, including ThingSpeak, Twilio, Adafruit I/O, api.OpenWeather.gov, imgur, query.yahooapis.com, Microsoft Azure IoT, IBM Watson IoT, Blynk, Twitter, and Spotify to name just a few.