Skywire® Nano NL-SW-LTE-NRF9160 AWS IoT with TLS

NimbeLink Corp
Updated: August 2020

Table of Contents

1. Introduction

1.1 Overview

This document serves as a guide for Amazon AWS IoT connections using the NimbeLink 4G CAT M1 NRF9160 Nano Skywire. AWS IoT requires Transport Layer Security (TLS) for device connections. This tutorial will document the configuration of the modem and the Amazon AWS settings, and will demonstrate two different connection examples.

1.2 Orderable Parts

Orderable Device	Description	Carrier	Network Type
NL-SW-LTE-NRF9160	4G LTE CAT M1	AT&T, Verizon	LTE

1.3 Disclaimer

Please note that the NL-SWN-LTE-NRF9160's built-in TLS stack does not support server name indication (SNI). SNI is a TLS extension where the client indicates the hostname it is trying to connect to as part of the TLS handshake. For more information, please see: https://tools.ietf.org/html/rfc6066

When using the modem's built-in TLS stack, your endpoint must not require SNI for a successful TLS connection. If your endpoint requires SNI, you will need to run your own SSL/TLS stack on your host processor and use the NL-SWN-LTE-NRF9160's TCP sockets. For example, mbed TLS (https://tls.mbed.org/) and wolfSSL (https://www.wolfssl.com/) are two external TLS stacks. See https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations for more alternatives. NimbeLink does not provide support for integrating these embedded libraries into your host platform.

Alternatively, a proxy server could be set up that does not require SNI, but can communicate with the server that does. NimbeLink does not provide support for setting this up.

2. AWS IoT Setup

2.1 Preliminary Setup

Before starting, it is important to note that this guide assumes that the reader already has a valid Amazon AWS account. If this is not the case, Amazon offers a free trial account that can be used to test this guide. For more information about the free account, please follow this link:

https://aws.amazon.com/free/

2.2 Create a Policy

The first step in the AWS connection process is to create a policy. Login to the AWS IoT console at the following link:

https://console.aws.amazon.com/iot/

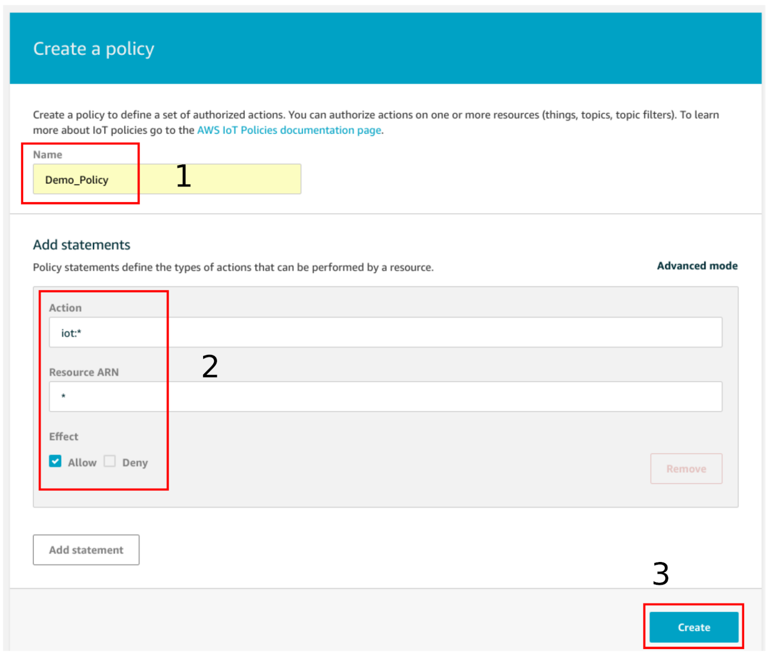
and navigate to the 'Secure' > 'Policies' menu. Once there, press the “Create a policy" button located near the center of the screen.

In the next page, choose a name (1.1) for the policy. Also choose “iot:*" for the “Action" (1.2) and “*" for the “Resource ARN" (1.2) fields. Check the “Allow" box (1.2), and then click "Add Statement". Finally, click "Create" (1.3) to create the policy. Refer to the image below.

Warning: This policy is very permissive, and AWS recommends following a least permissive approach to policies. Example policies that follow this practice can be found at the following resources if you’re interested in learning more:

HTTP:
https://docs.aws.amazon.com/iot/latest/developerguide/device-shadow-rest-api.html?icmpid=docs_iot_console

MQTT:
https://docs.aws.amazon.com/iot/latest/developerguide/device-shadow-mqtt.html?icmpid=docs_iot_console


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). To learn more about IoT policies go to the AWS IoT Policies documentation page.

Name: Demo_Policy

Add statements

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

Action: iot:*

Resource ARN: *

Effect: [✓ Allow] Deny

Figure 1. Create a Policy

2.3 Create a "Thing"

Next, navigate to 'Manage' > 'Things' (2.1) using the menu on the left-hand side of the dashboard. Next, select ‘Create’ (2.2) in the top right corner to make a new “thing". Refer to the image below for reference.

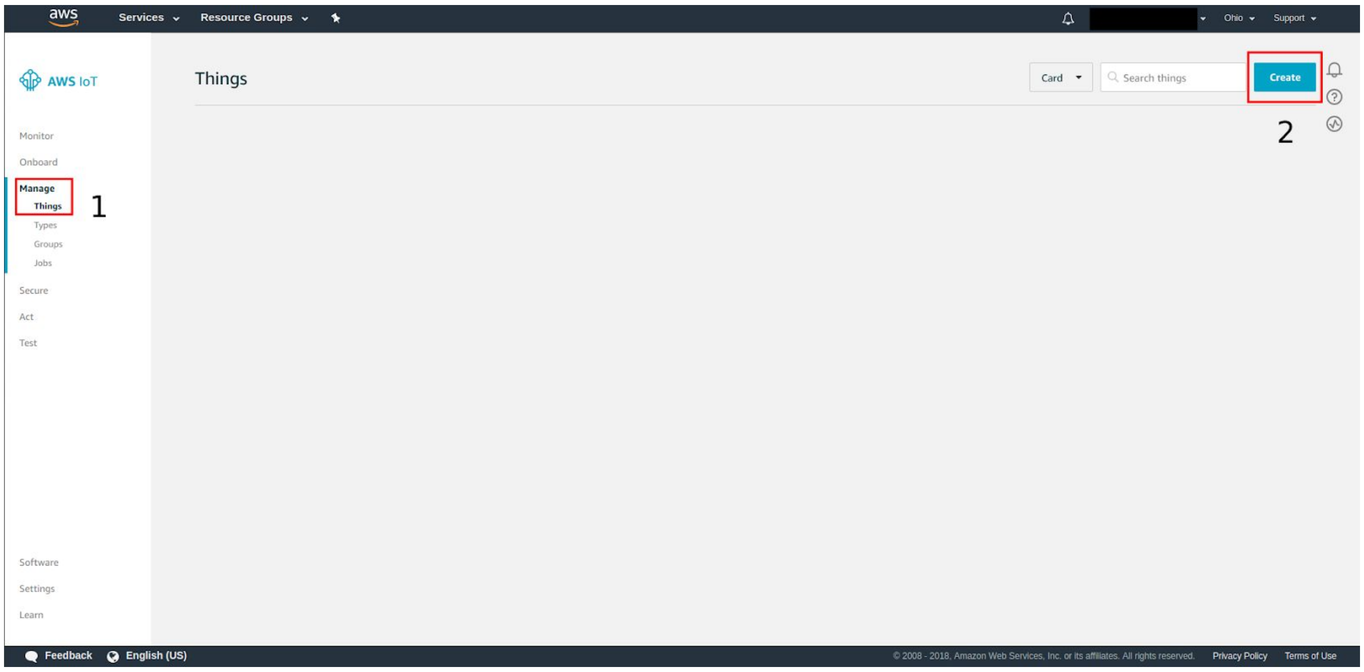
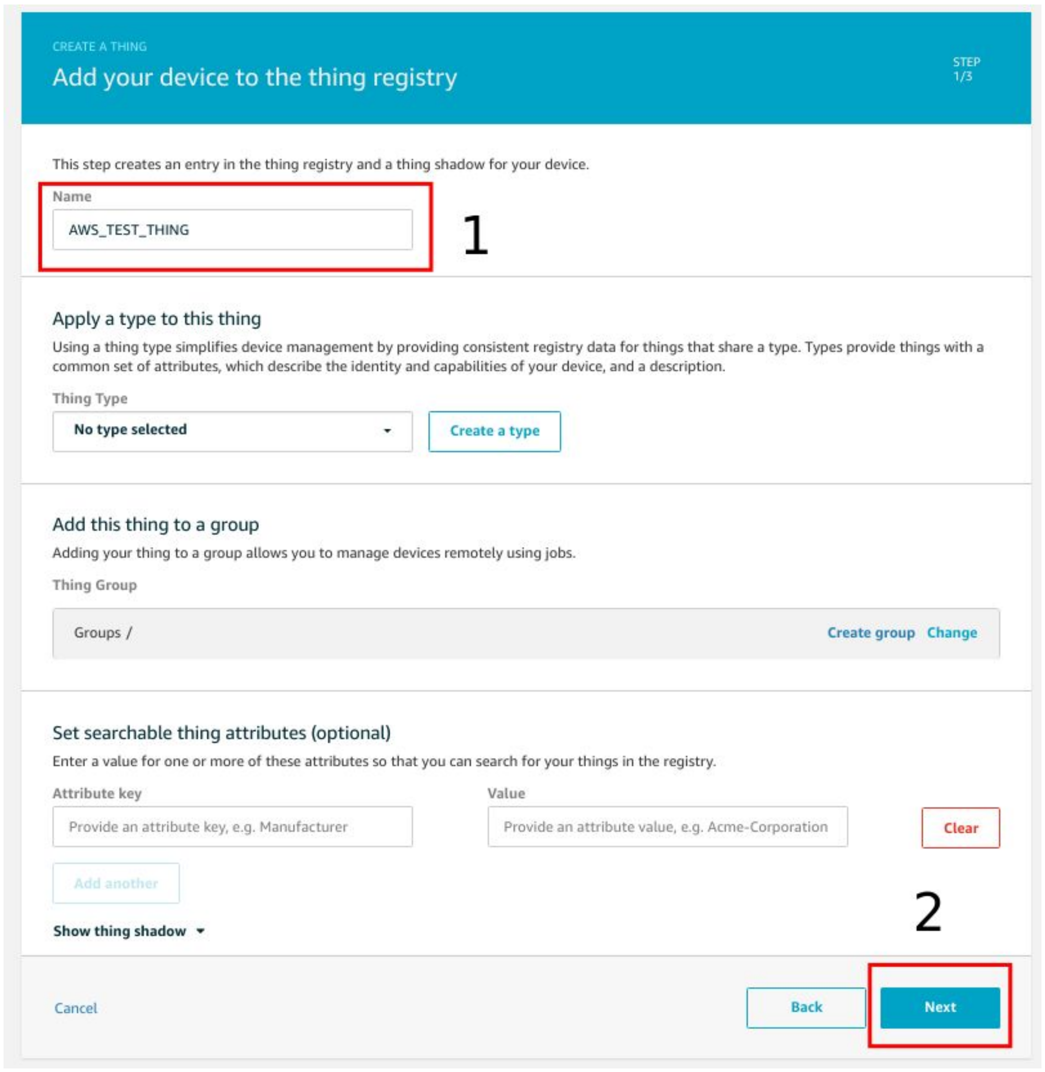


Figure 2. Thing Management Page

After pressing the “Create" button (2.2), select the “Create a single thing" option in the next page that loads. In the following page, enter a custom name in the appropriate box (3.1), and then press the “Next" button (3.2). The webpage should look something like this:



CREATE A THING

Add your device to the thing registry

STEP 1/3

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

Name: AWS_TEST_THING

Apply a type to this thing

Using a thing type simplifies device management by providing consistent registry data for things that share a type. Types provide things with a common set of attributes, which describe the identity and capabilities of your device, and a description.

Thing Type: No type selected

Add this thing to a group

Adding your thing to a group allows you to manage devices remotely using jobs.

Thing Group: Groups /

Set searchable thing attributes (optional)

Enter a value for one or more of these attributes so that you can search for your things in the registry.

Attribute key: Provide an attribute key, e.g. Manufacturer

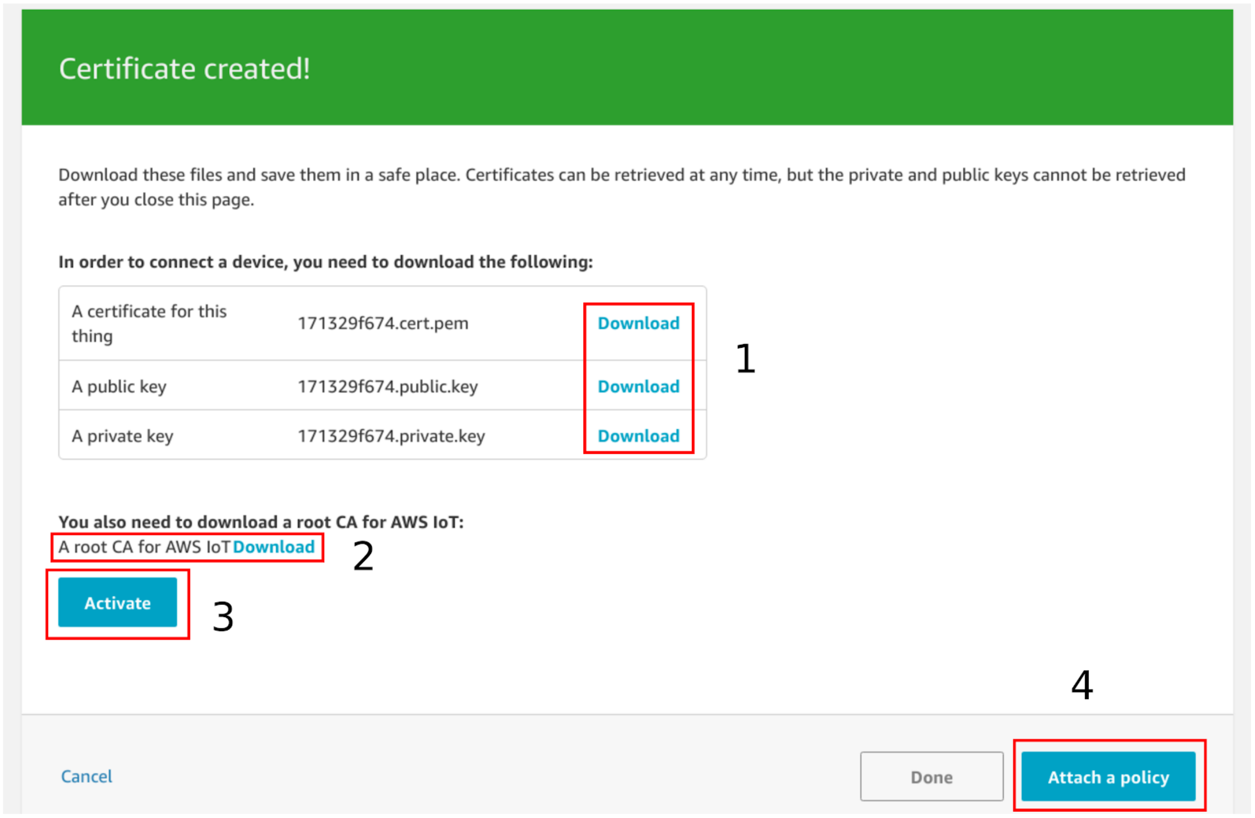
Value: Provide an attribute value, e.g. Acme-Corporation

Figure 3. Creating a Thing

2.4 Generate Certificates

After pressing the “Next" button (3.2), select the “Create certificate" option in the next web page that loads. Amazon AWS will then generate a client certificate, private key, and a public key for the “thing" that was just created. Download these certificates (4.1) and save them in a convenient place. Also, be sure to download the Amazon AWS CA certificate (4.2) as this will be needed for the TLS connection.

Next, press the “Activate" button (4.3) to assign the generated certificates to the “thing". Finally, click “Attach a policy" (4.4) to proceed to the next step. Refer to the image below for reference.



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	171329f674.cert.pem	Download
A public key	171329f674.public.key	Download
A private key	171329f674.private.key	Download

You also need to download a root CA for AWS IoT:

A root CA for AWS IoT Download

Figure 4. Creating Certificates

Note: The public and private key can only be downloaded from this page. Once this page is navigated away from, these files will no longer be available for download.

2.5 Attach the Policy to the "Thing"

After advancing to the next page, attach the policy (5.1) created in Section 2.2 to the “thing" created in Section 2.3 and click “Register Thing” (5.2). Refer to the image below as an example.

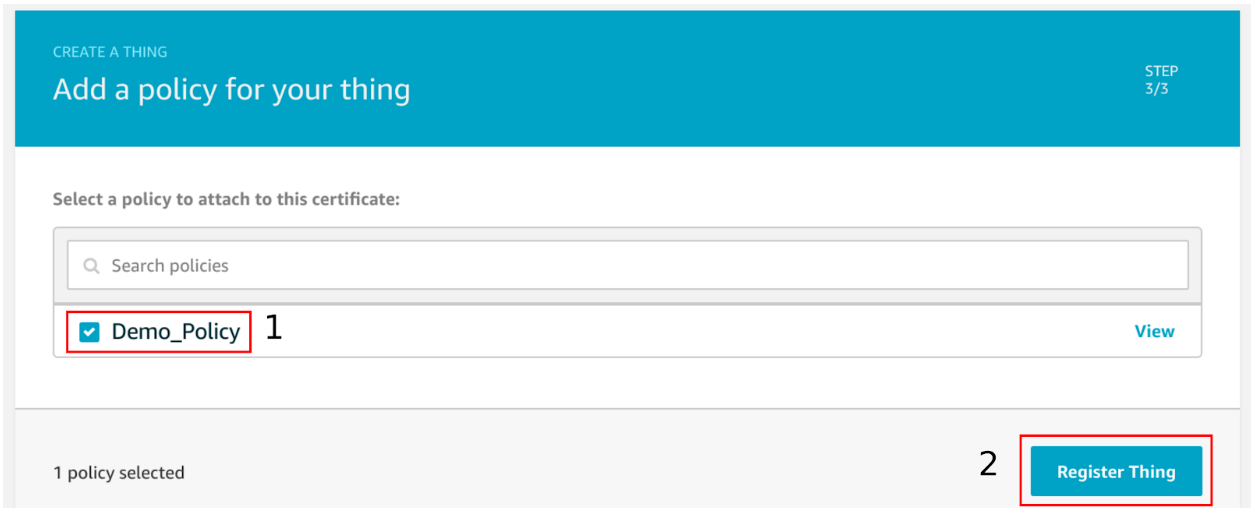


Figure 5. Attaching Policy to Thing

After each of the steps in Section 2 have been completed, proceed to Section 3 for the Skywire configuration instructions.

3. Skywire Configuration

3.1 Prerequisites

This section assumes that the three certificates from AWS IoT Console have been downloaded, the modem is set up and registered on the network, and the CME Errors are set to verbose (`AT+CMEE=2`)

3.2 Uploading Certificates

The first step in the configuration of the Skywire is to upload the certificates needed for the TLS connection. These three certificates are the private key, client certificate and the CA certificate.

The next two sections will lay out instructions for uploading the certificates in a Linux and a Windows environment. Section 3.2.1 details the Linux instructions, while Section 3.2.2 contains the Windows instructions.

3.2.1 Certificate Uploading Using a Linux Environment

First open a terminal and navigate to the directory that contains the certificates that were downloaded in Section 2.4.

Next, establish a connection to the Nano using a preferred serial console. See the Skywire Nano User Manual for more information on connecting to the nano.

Once the serial console has been set up properly, issue the following command to put the Nano in airplane mode so certificates can be uploaded to it.

```
AT+CFUN=4
```

Now we will load the certificates onto the Nano.

1. Type in the AT command below, where **`x`** is the sec_tag (any number in **`[0,2147483648]`**, though we will use **`3, 4,`** and **`5`** in our example later) and **`y`** is replaced with **`0, 1,`** or **`2`** depending on if the certificate being uploaded (rootCA, device certificate, private key, respectively). **Do not hit enter.**

   ```
   AT%CMNG=0,x,y,"
   ```

2. We need to transfer the cert exactly as it is from the computer to the Nano. The easiest way is described below.
3. First, close the connection between the serial port and the terminal. Then, issue the following command in the Linux terminal, where the name of the file is replaced with the name of the certificate in question, and the destination is replaced with the appropriate path to the serial line:

```
cat VeriSign-Class\3-Public-Primary-Certification-Authority-G5.pem > /dev/ttyACM1
```

This command will pipe the contents of the certificate to the serial line, which will then be stored in a file on the Nano. Reopen the serial connection with the modem, **enter the closing quote**, and press **Enter** to finish uploading.

The serial line will respond with `OK`

Repeat the three steps above until each of the three files have been uploaded. Once the files are confirmed to have been uploaded successfully, proceed to Section 3.2.

3.2.2 Certificate Uploading Using a Windows Environment

First open the Windows command prompt and navigate to the directory that contains the certificates that were downloaded in Section 2.4. Type "**`dir`**" to list the contents of the directory on individual lines. Take note of the file sizes of each of the relevant certificates. This information will be needed shortly.

Next, establish a connection to the Nano using a preferred serial console. Once the serial console has been set up properly, issue the following command to put the Nano in airplane mode so certificates can be uploaded to it.

```
AT+CFUN=4
```

Now we will load the certificates onto the Nano.

1. Type in the AT command below, where **`x`** is the sec_tag (any number in **`[0,2147483648]`**, though we will use **`3, 4,`** and **`5`** in our example later) and **`y`** is replaced with **`0, 1,`** or **`2`** depending on the certificate being uploaded (rootCA, device certificate, private key, respectively). **Do not hit enter.**

   ```
   AT%CMNG=0,x,y,"
   ```

2. We need to transfer the cert exactly as it is from the computer to the Nano. The easiest way is described below.
3. First, close the connection between the serial port and the terminal emulator. Next, issue the following command in the command prompt, where the name of the file is replaced with the name of the certificate being uploaded, and the "`COM10`" string is replaced with the proper COM port number.

   ```
   copy 8da6fe87f3-certificate.pem \\.\COM10
   ```

   This command will pipe the contents of the certificate to the serial line, which will then be stored in a file on the Nano. Reopen the serial connection with the modem, **enter the closing quote**, and press **Enter** to finish uploading.

   The serial line will respond with **`OK.`**

Repeat the three steps listed above until each of the three files have been uploaded. After the files have been successfully uploaded, reattach the serial console to the appropriate COM port. Once finished, proceed to Section 3.2.

3.3 Verifying the Certificate Uploads

Assume the files were uploaded to indices **3, 4,** and **5**, as in our example in section 4. Again, the **0, 1,** and **2** at the end of the command represent the type of the certificate (rootCA, device certificate, private key, respectively). Verify that the rootCA upload was successful by issuing the following command:

```
AT%CMNG=2,3,0
```

Note: you cannot read back the device certificate (index 4) nor the private key (index 5) once entered. Thus, the commands **AT%CMNG=2,4,0** and **AT%CMNG=2,5,0** will fail.

The terminal should respond with something similar to the following:

```
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB
.                             .                                .
.                             .                                .
.                             .                                .
hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
-----END CERTIFICATE-----
OK
```

It is possible for the files to have different formatting and not print exactly like above.

After each of the three files have been uploaded, reconnect to the network with **AT+CFUN=1** and proceed to Section 3.4.

3.4 TLS Profile Configuration

The next step is to configure the TLS socket on the Skywire. To do this, issue the following command:

```
AT#XTLSSOCKET=<sock>,1,<host_url>,<sec_tag>[,<sec_tag>[,...]]
```

Where **<sock>** is the socket, which is an integer, **<host_url>** is the endpoint to connect to, and **<sec_tag>[,<sec_tag>[,...]]** are the tags associated with the certificates uploaded in Section 3.2.

For instance, the following command opens socket 1 to an Amazon AWS IoT Endpoint:

```
AT#XTLSSOCKET=1,1,"axxx....amazonaws.com",3,4,5
```

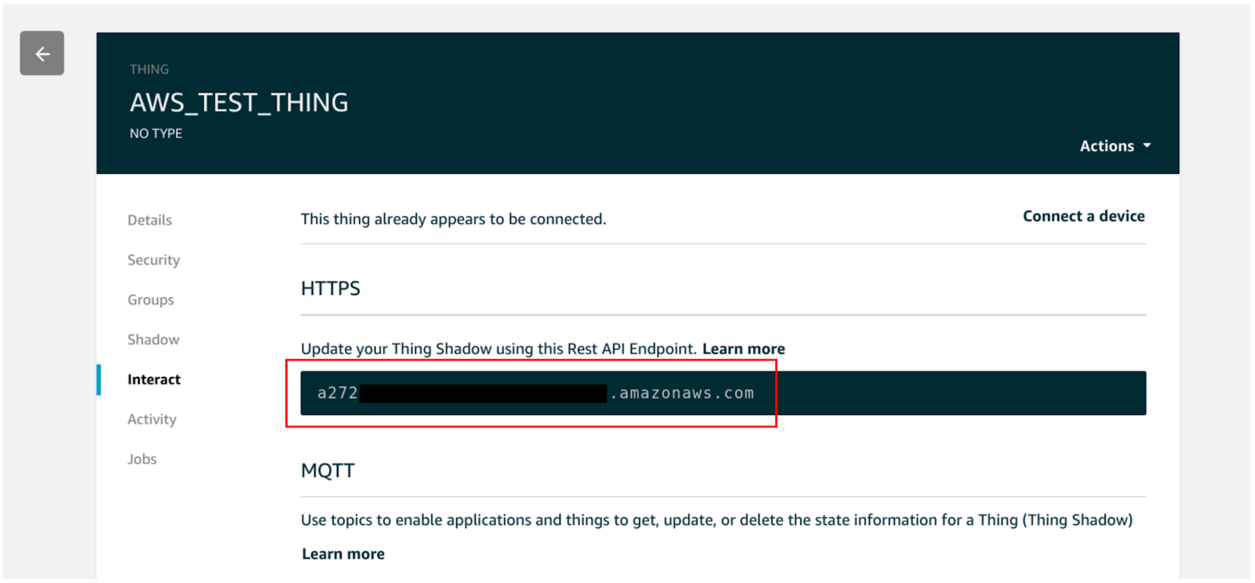
Once the above have been successfully completed, proceed to Section 4.

4. Connect to Amazon AWS

4.1 Connect to AWS Endpoint

After the Skywire has been configured properly, it is ready to establish a connection to the AWS server.

First, find the endpoint for the “thing” that was created on the AWS website. To do this, navigate to the “Things” page using the menu on the left-hand side of the AWS console page. Click on the “thing” and then navigate to the “Interact” menu. The correct menu should look something like this:



This thing already appears to be connected.

Update your Thing Shadow using this Rest API Endpoint. **Learn more**

```
a272[redacted].amazonaws.com
```

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

In the image above, the URL for the device endpoint has been enclosed in a red rectangle. Record whatever URL shows up in this page, as it will be needed in the TLS socket connection command.

To open the TLS socket, issue the command below. Make sure to replace the device endpoint in the AT command below with the endpoint that is unique to the AWS account being used for this example.

```
AT#XTCPCONN=1,"axxx...amazonaws.com",8443
```

Where **1** specifies the TLS socket to use (configured in section 3.4) and **8443** is the port to use.

After issuing this command, the modem will attempt to connect to the AWS endpoint. If the connection is successful, the modem will return the following after some time:

```
OK
```

The above text indicates that the TLS handshake was successful and that the socket has been connected.

After the TLS socket has been successfully connected. Proceed to Section 4.2.

4.2 Sending an HTTPS Request

After successfully opening an TLS socket, use the following command to send an HTTP request to the AWS endpoint:

```
AT#XTCPSEND=1
```

Where "**1**" is the number of the socket used for the TLS connection.

After issuing this AT command, the modem will wait for text to be entered into the terminal. This text can either be pasted or typed into the terminal. After all of the text has been entered or pasted in, press "**CTRL-Z**" to finalize the transmission. See the text below for a sample HTTP POST request using the "**AT#XTCPSEND**" command.

Also, take note of the text in red in the table on the next page. Any of the red text indicates key presses, and are not to be typed explicitly. These key press sequences are crucial in order to format the POST command properly. Finally, bold text signifies commands issued to the modem, and text pasted into the terminal.

```
AT#XTCPSEND=1
> POST /things/AWS_TEST_THING/shadow HTTP/1.1 CTRL+M CTRL+J
Host: axxx...amazonaws.com:8443 CTRL+M CTRL+J
Content-Type: application/json CTRL+M CTRL+J
Content-Length: 123 CTRL+M CTRL+J CTRL+M CTRL+J

{"state":{"desired":{"string1":"TLS Connect to AWS","string2":"Using the
built-in stack","string3":"of the Skywire Nano"}}}
CTRL-Z
#XTCPSEND: 331


OK
SOCK: 1,DTA
```

As can be seen in the text above, the POST command was entered into the terminal, and the modem responded with "**`OK`**" indicating that the transmission succeeded. Also note that the "**`SOCK: 1,DTA`**" URC indicates that an HTTP response was received. Section 4.3 will detail how to read this response.

Another important item to note is the "**`Content-Length: 116`**" line. In this case, the value of "**`116`**" indicates that 116 bytes of data are being sent through the socket. This helps the endpoint know how many bytes to consider as data. It is crucial to ensure that this number is updated whenever the data JSON is edited.

Note: Properly formatting the POST command can be challenging. Below are a few pointers for correct POST command formatting and entry:

- It is usually impossible to paste the entire POST command all at once. Try pasting the command in line-by-line as opposed to all at once.
- Press **`CTRL+M CTRL+J`** after each line of the POST command. This sequence enters in a newline and carriage return character after each line.
- Press **`CTRL+M CTRL+J CTRL+M CTRL+J`** after the "**`Content-Length: 116`**" line. In other words, insert two newline and carriage return sequences after this line.
- Replace the "**`AWS_TEST_THING`**" identifier with the unique name assigned during the "thing" creation in Section 2.3.
- Replace the AWS endpoint "**`axxx...amazonaws.com:8443`**" with the unique endpoint associated with the Amazon AWS account in use.
- If the contents of the JSON are changed for any reason, the integer argument of the "**`Content-Length: 116`**" line must be modified.

- In other words, if the JSON is made larger or smaller, the total number of bytes being sent must be recalculated, and the "**`Content-Length: x`**" line must be updated with the new length.

4.3 Reading an HTTPS Response

To read an HTTP response, issue the following command:

`AT#XTCPRECV=1,500,10`

Where "**`1`**" is replaced by the number of the socket used for the TLS connection, **`500`** is replaced by the number of bytes to read, and **`10`** is the timeout.

In the case of the sample POST command in Section 4.2, the HTTP response follows.

```
AT#XTCPRECV=1,500,10
SSLRECV: 488,HTTP/1.1 200 OK
content-type: application/json
content-length: 72
date: Wed, 24 Jun 2020 19:59:48 GMT
x-amzn-RequestId: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
connection: keep-alive
<message>

OK
```

Note that multiple **`XTCPRECV`** statements can be sent. For instance, the output could look like the following:

```
AT#XTCPRECV=1,50,10
SSLRECV: 50,HTTP/1.1 200 OK
content-type: application/

OK


AT#TCPRECV=1,433,10
json
content-length: 72
date: Wed, 24 Jun 2020 19:59:48 GMT
x-amzn-RequestId: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
connection: keep-alive
<message>

OK
```

Note that the max length readable at one time is **`1000`** bytes.

4.4 Closing a TLS Socket

To close an TLS socket, issue the following command:

`AT#XTLSSOCKET=1,0`

Where "**`1`**" is replaced with the number of the socket that was used for the TLS connection. The modem should respond with **`OK`**, unless the connection was already terminated by the server. In this case, the modem will respond with an **`ERROR`**

5. Working Example: AWS IoT

This section is split up into two subsections. The first shows the preliminary set up of the TLS socket and the second shows communication with AWS IoT. For AWS IoT, three certs are needed: the rootCA, client certificate, and client private key.

Comments are in // blue

Note: Minicom interprets CRLF differently, which is why the certs and HTTP requests are formatted in an odd manner. The certs and HTTP requests were written into the serial line using the **`cat`** method, described above in Section 3.2.

5.1 Set Up

```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Aug 13 2017, 15:25:34.
Port /dev/ttyACM1, 09:48:48

Press CTRL-A Z for help on special keys

AT#CFUN=4 // airplane mode
OK
// Write the rootCA
AT%CMNG=0,3,0,"-----BEGIN CERTIFICATE-----

MIIDWTCCAkGgAwIBAgIUJ+1lL9xBBn64vmI4yU9JRHZKBGYwDQYJKoZIhvcNAQE
L

BQAwTTFLMEkGA1UECwxCQW1hem9uIFdlYiBTZXJ2aWNlcyBPPUFtYXpvbi5jb20g

SW5jLiBMPVNlYXR0bGUgU1Q9V2FzaGluZ3RvbiBDPVVTMB4XDTIwM1
...

-
"
OK
// Write the client cert
AT%CMNG=0,4,1,"-----BEGIN CERTIFICATE-----
```

```
...
“
OK
// Write the client private key
AT%CMNG=0,5,2,"-----BEGIN RSA PRIVATE KEY----
...
“
OK
AT+CFUN=1
OK
```

5.2 Communication

```
// open SSL socket to unique endpoint
AT#XTLSSOCKET=1,1,"axxx....amazonaws.com",0,1,2
#XTLSSOCKET: 1,258
OK
AT#XTCPCONN=1,"axxx....amazonaws.com",8443
#XTCPCONN: 1
OK
AT#XTCPSEND=1 // Send data over socket
> GET /things/AWS_TEST_THING/shadow HTTP/1.1 // HTTP request to get the
current shadow

CTRL+M CTRL+J

CTRL+M CTRL+J CTRL+Z
#XTCPSEND: 37 // sent 37 bytes

OK
SOCK: 1,DTA // received data
AT#XTCPRECV=1,500,10
#XTCPRECV: 488,HTTP/1.1 200 OK
content-type: application/json
content-length: 299
date: Wed, 01 Jul 2020 17:59:05 GMT
x-amzn-RequestId: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
connection: keep-alive

{"state":{"desired":{"color":"blue","power":"off"},"reported":{"color":"blue","power":"off"}}
,"metadata":{"desired":{"color":{"timestamp":1593096284},"power":{"timestamp":15930
96284}},"reported":{"color":{"time}
OK
```

```
AT#XTCPSEND=1 // send data over socket
// this was inputted from a file with the cat/copy method described in section 3.2
> POST /things/AWS_TEST_THING/shadow HTTP/1.1 // HTTP request to update the
shadow
                              Host: axxx...amazonaws.com
                                                                      Content-Type:
application/json

Content-Length: 53


{"state":{"reported":{"color":"blue","power":"off"}}} CTRL+Z

OK

SOCK: 1,DTA // received data
AT#XTCPRECV=1,369,10 // read 369 bytes from the socket
#XTCPRECV: 369,HTTP/1.1 200 OK
content-type: application/json
content-length: 180
date: Thu, 25 Jun 2020 15:21:23 GMT
x-amzn-RequestId: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
connection: keep-alive

{"state":{"reported":{"color":"blue","power":"off"}},"metadata":{"reported":{"color":{"time
stamp":1593098483},"power":{"timestamp":1593098483}}},"version":4,"timestamp":15
93098483}

OK
AT#XTCPSEND=1 // send data over the socket
> GET /things/AWS_TEST_THING/shadow HTTP/1.1 // HTTP request to get the
current shadow
CTRL+M CTRL+J
CTRL+M CTRL+J CTRL+Z

OK

SOCK: 1,DTA // received data
AT#XTCPRECV=1,488,10 // read 488 bytes from the socket
#XTCPRECV: 488,HTTP/1.1 200 OK
content-type: application/json
content-length: 299
date: Thu, 25 Jun 2020 15:21:42 GMT
```

```
x-amzn-RequestId: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
connection: keep-alive

{"state":{"desired":{"color":"blue","power":"off"},"reported":{"color":"blue","power":"off"}}
,"metadata":{"desired":{"color":{"timestamp":1593096284},"power":{"timestamp":15930
96284}},"reported":{"color":{"timestamp":1593098483},"power":{"timestamp":1593098
483}}},"version":4,"timestamp":1593098502}

OK
AT#XTLSSOCKET=1,0 // close the socket
#XTLSSOCKET: 0
OK
```

6. Troubleshooting

6.1 HTTP Response Codes

6.1.1 403 Forbidden

If a connection can be established, but the AWS response to the "**`GET`**" command is "**`403 Forbidden`**", make sure that the current AWS policy is set to allow all IoT actions. This can be done through the AWS IoT Console.

6.1.2 400 Bad Request

If a connection can be established, but the AWS response to the "**`GET`**" or "**`POST`**" command is "**`400 Bad Request`**", make sure the syntax of the "**`GET`**" or "**`POST`**" command is correct.

6.1.3 404 Not Found

If a connection can be established, but the AWS response to the "**`GET`**" or "**`POST`**" command is "**`404 Not Found`**", the HTTP request cannot be fulfilled because the resource does not exist. For instance, there might be a typo, like

```
GET /things/TCNAG/shadow HTTP/1.1
```

instead of

```
GET /things/AWS_TEST_THING/shadow HTTP/1.1
```

6.2 Verify Credentials

If for some reason the credentials for the AWS connection do not work, OpenSSL can be used to check their validity. This process is helpful for narrowing down the source of the connection issue.

6.2.1 Testing AWS Credentials using OpenSSL

To test credentials with OpenSSL, first ensure that OpenSSL is properly installed on a Linux or Windows system. Next, navigate to the directory that contains the certificates that are being tested. Issue the following command to attempt a connection to AWS:

```
openssl s_client -servername axxx...amazonaws.com -connect
axxx...amazonaws.com:8443 -CAfile VeriSign-Class\
3-Public-Primary-Certification-Authority-G5.pem -cert
8da6fe87f3-certificate.pem.crt -key 8da6fe87f3-private.pem.key -certform PEM
-keyform PEM
```

Be sure to replace any of the text in bold with unique certificate names, and the unique AWS endpoint URL.

If the connection is successful, the terminal should respond with "`CONNECTED`" followed by some information about the connection, including the server certificate. To further test the connection, type the following command, followed by the sequence "**`CTRL+M, CTRL+J`**":

```
GET /things/AWS_TEST_THING/shadow HTTP/1.1
```

Remember to replace the "**`AWS_TEST_THING`**" string with the unique "thing" name assigned in Section 2.3.

The server should respond with something similar to:

```
HTTP/1.1 200 OK

content-type: application/json

content-length: 385

date: Tue, 24 Jul 2018 22:35:57 GMT

x-amzn-RequestId: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

connection: keep-alive


{"state":{"desired":{"string1":"TLS Connect to AWS","string2":"Using
the built-in stack","string3":"of the Nano"},"delta":{"string1":"TLS
Connect to AWS","string2":"Using the built-in stack","string3":"of
the
Nano"}},"metadata":{"desired":{"string1":{"timestamp":1532463231},"st
ring2":{"timestamp":1532463231},"string3":{"timestamp":1532463231}}},
"version":19,"timestamp":1532471757}
```

If a valid connection can be established, then it is safe to say that the certificates are indeed valid, and thus are not the source of the problem.