AWS IoT Example Using the Raspberry Pi 3 and NimbeLink CAT M1 Skywire®

NimbeLink Corp

Updated: August 2017

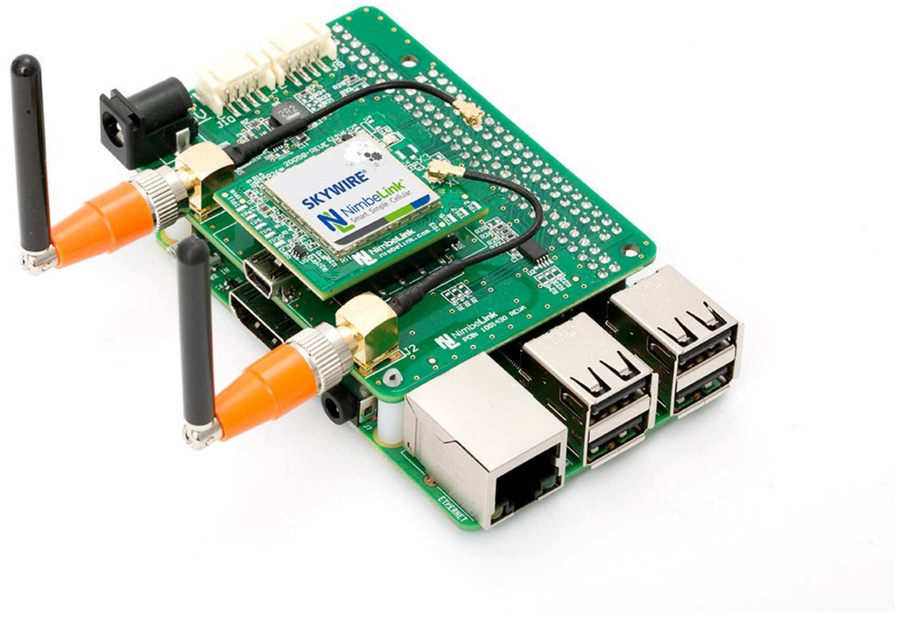

Table of Contents

1. Introduction

1.1 Orderable Part Numbers

Orderable Device	Description	Carrier	Network Type
NL-AB-RPI	Skywire Raspberry Pi Hat	Any	Any
NL-SW-LTE-SVZM20-ES	LTE CAT M1, Engineering Sample	Verizon	LTE
NL-SW-LTE-SVZM20	LTE CAT M1	Verizon	LTE

1.2 Overview

Amazon's AWS IoT provides a powerful and easy-to-use IoT management platform. Whether you want to connect a single device, or deploy a fleet of devices, AWS IoT provides an excellent option for usage.

This example provides an overview for using a Raspberry Pi 3, connected to a NimbeLink Raspberry Pi Hat (NL-AB-RPI) and a NimbeLink CAT M1 Skywire®.

Note: Though we are using a NimbeLink CAT M1 Skywire embedded cellular modem in this example, this guide is easily adaptable to use any of our Skywire modems with their respective data connection (PPP, CDC-Ether, QMI, etc.).

1.3 Prerequisites

This application note assumes you have some familiarity with the Raspberry Pi 3, including how to load an image of Raspbian on your Raspberry Pi, how to connect a monitor, keyboard, and mouse to your Raspberry Pi, how to setup an Ethernet and Wifi connection, and basic familiarity with the Linux command line on the Raspberry Pi. If you don't, please consult the "Help" section from the Raspberry Pi Foundation's website and follow the guides there:

https://www.raspberrypi.org/help/

If you are unable to connect a keyboard, mouse, and monitor to your Raspberry Pi, you can remotely connection to the Raspberry Pi using either a Serial connection or SSH.

In addition, this guide requires downloading files from Amazon AWS, so you will need an Ethernet connection or Wifi connection for the initial setup of the Raspberry Pi.

Note: We recommend that you do not use the cellular connection to do the downloads mentioned in this document. If performed using the cellular connection these downloads will count against your data usage.

Finally, this guide assumes that you have verified that your CAT M1 Skywire is working and transmitting data, and that PPP working on your Raspberry Pi 3, NL-AB-RPI, and NL-SW-LTE-SVZM2x Skywire. If you have not done so already, please complete the following walkthroughs and application notes:

NimbeLink Skywire Raspberry Pi Adapter User Manual

http://nimbelink.com/Documentation/Development_Kits/NL-AB-RPI/1001464_NL-AB-RPI_UserManual.pdf

Using PPP on the Raspberry Pi with CAT M1 Skywire

http://nimbelink.com/Documentation/Skywire/4G_LTE_Cat_M1/30273_NL-SW-LTE-SVZM21_PPP-on-CATM1_RPi.pdf

Note: Though this guide is written using the NL-SW-LTE-SVZM20 Skywire, you can use the data connection on whichever Skywire you would like. For example, here are guides for our LTE CAT1 Skywires:

NL-SW-LTE-GELS3 Skywire Family

CDC-Ether:
http://nimbelink.com/Documentation/Skywire/4G_LTE_Cat_1/30111_NL-SW-LTE-GELS3_BBB_CDC-ETHER.pdf

PPP:
http://nimbelink.com/Documentation/Skywire/4G_LTE_Cat_1/30166_NL-SW-LTE-GELS3_PPP.pdf

NL-SW-LTE-WM14 Skywire Family

CDC-Ether:
http://nimbelink.com/Documentation/Skywire/4G_LTE_Cat_1_WNC/30262_NL-SW-LTE-WM14_CDC-ETHER.pdf

PPP: http://nimbelink.com/Documentation/Skywire/1001463_Document-Guide.pdf

If your Skywire is not listed above, please see the product page for your Skywire to find the correct Application Note for PPP or CDC-ETHER/NCM/QMI connections.

1.4 Testing

This guide has been tested on the following hardware and operating systems:

Hardware

Raspberry Pi 3 Model B

Operating Systems

Raspbian 8.0 Jessie Kernel 4.4.50-v7+

Raspbian 8.0 Jessie Kernel 4.9.41-v7+

2. AWS IoT Example

2.1 Amazon's AWS IoT Guide

Amazon has written an excellent guide for getting started with AWS IoT. We will be following that guide, with a few additional steps added in to have this work using the cellular network with the Skywire modem.

2.2 AWS IoT SDK Tutorials

Amazon's AWS IoT Guide starts with an introduction, located here:

http://docs.aws.amazon.com/iot/latest/developerguide/sdk-tutorials.html

Open the link on your Raspberry Pi (or workstation, that is connected to your Raspberry Pi), and read the introduction. Once completed, click on the first link under "Contents", which is "Connecting Your Raspberry Pi":

http://docs.aws.amazon.com/iot/latest/developerguide/iot-sdk-setup.html

2.3 Connecting Your Raspberry Pi

Follow this guide to create your Amazon AWS IoT account, register a Thing, and get certificates:

http://docs.aws.amazon.com/iot/latest/developerguide/iot-sdk-setup.html

Helpful Tip: The guide talks about a "root CA certificate". You will download that certificate on the next step.

Once you have completed the guide on this page, click "Next >>" at the bottom of the page, or click "Using the AWS IoT Embedded C SDK" link.

2.4 Using the AWS IoT Embedded C SDK

Follow this guide to download the AWS IoT Embedded C SDK, move the certification files to the proper location, setup the subscribe_publish_sample example:

http://docs.aws.amazon.com/iot/latest/developerguide/iot-embedded-c-sdk.html

Stop following the guide at the step "Run Sample Applications".

2.5 Setup PPP Connection

At this point, you can use the Skywire modem’s PPP connection to send data to AWS IoT. Disconnect your Ethernet or Wifi connection, and start your PPP connection according to this guide for the Skywire M1 modem:

http://nimbelink.com/Documentation/Skywire/4G_LTE_Cat_M1/30273_NL-SW-LTE-SVZM21_PPP-on-CATM1_RPi.pdf

2.6 Run Sample Applications

Once you have established your PPP connection, continue following Amazon's AWS IoT guide at the place you left off, "Run Sample Applications":

http://docs.aws.amazon.com/iot/latest/developerguide/iot-embedded-c-sdk.html

3. AWS IoT Shadow Sample

3.1 Introduction

Now that you have a PPP connection using the NimbeLink Skywire modem, we will walk through another example for AWS IoT. In this example, we will send data to a Thing Shadow. For more information about Shadows, please see the below page:

http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

Note: Section 3 assumes you have completed Section 2. If you have not, please complete Section 2.

3.2 Copy AWS IoT Config File

If you are not already, navigate to the following directory:

```
$ cd [path to deviceSDK]/sample_apps/shadow_sample/
```

Note: This is the directory you ended up in at the end of Section 2.

Copy the `aws_iot_config.h` file to the example shadow_sample directory:

```
$ cp aws_iot_config.h ../shadow_sample/
```

3.3 Navigate to the shadow_sample Directory

Next, navigate to the shadow_sample directory:

```
$ cd ../shadow_sample/
```

3.4 Compile and Run the Program

Use make to compile to program:

```
$ make -f Makefile
```

Now run the program:

```
$ ./shadow_sample
```

The program will send information to your AWS IoT Dashboard. You should see something similar to the following:

```
AWS IoT SDK Version(dev) 1.1.2-

DEBUG:   main L#181 Using rootCA
/home/pi/aws/deviceSDK/sample_apps/shadow_sample/../../certs/VeriSign-Class
3-Public-Primary-Certification-Authority-G5.pem
DEBUG:   main L#182 Using clientCRT
/home/pi/aws/deviceSDK/sample_apps/shadow_sample/../../certs/3640a6fd11-certificate.pem.crt
DEBUG:   main L#183 Using clientKey
/home/pi/aws/deviceSDK/sample_apps/shadow_sample/../../certs/3640a6fd11-private.pem.key
Shadow Init
Shadow Connect
DEBUG:   aws_iot_shadow_connect L#77 Thing Name MyRaspberryPi
DEBUG:   aws_iot_shadow_connect L#78 MQTT Client ID MyRaspberryPi
DEBUG:   registerJsonTokenOnDelta L#104 delta topic $aws/things/MyRaspberryPi/shadow/update/delta

=======================================================================================

On Device: window state false
Update Shadow: {"state":{"reported":{"temperature":25.500000,"windowOpen":false}},
"clientToken":"MyRaspberryPi-0"}
*****************************************************************************************

Update Accepted !!

=======================================================================================

On Device: window state false
Update Shadow: {"state":{"reported":{"temperature":26.000000,"windowOpen":false}},
"clientToken":"MyRaspberryPi-1"}
*****************************************************************************************

Update Accepted !!

=======================================================================================

On Device: window state false
Update Shadow: {"state":{"reported":{"temperature":26.500000,"windowOpen":false}},
"clientToken":"MyRaspberryPi-2"}
*****************************************************************************************

Update Accepted !!

=======================================================================================

On Device: window state false
Update Shadow: {"state":{"reported":{"temperature":27.000000,"windowOpen":false}},
"clientToken":"MyRaspberryPi-3"}
*****************************************************************************************
```

4. AWS IoT Python SDK Example: Basic PubSub

4.1 Introduction

Amazon has an SDK available for Python, as well. In this example, we will walkthrough the same example outlined in Section 2.

Note: this guide assumes that you have completed Section 2 at least through Section 2.3, and have completed this guide:

http://docs.aws.amazon.com/iot/latest/developerguide/iot-sdk-setup.html

In addition, this guide assumes that you have the root CA certificate. If you do not, please follow the Python README to get your root CA certificate:

https://github.com/aws/aws-iot-device-sdk-python/blob/master/README.rst

4.2 Bring Down PPP Connection

If you have the Skywire modem PPP connection still up, bring it down to download the AWS IoT Python SDK. To do this, issue:

```
$ sudo poff vzw-SVZM2x
```

Once your PPP interface is down, remove the USB connection and power cable from the development kit.

4.3 Bring Up Ethernet or Wifi Connection

Attach your Ethernet cable or setup your Wifi connection.

4.4 Download and Install AWS IoT Python SDK

Navigate to your preferred download directory. This example will navigate to the `pi` user directory:

```
$ cd /home/pi/
```

Once you are in your preferred directory, download and install the AWS IoT Python SDK. There are multiple methods to do this, depending on your preference. Please see the AWS IoT Python SDK README for detailed information:

https://github.com/aws/aws-iot-device-sdk-python/blob/master/README.rst

This example will use the "Build from source" method, which required cloning the Github repo and installing the Python SDK.

First, make sure you have git installed:

```
$ sudo apt-get update && sudo apt-get install git
```

Next, clone the repo:

```
$ git clone https://github.com/aws/aws-iot-device-sdk-python.git
```

Once the download completes, navigate to that directory:

```
$ cd aws-iot-device-sdk-python
```

Finally, install the SDK:

```
$ python setup.py install
```

Note: You may receive errors about not having permissions to install the SDK. If that is the case, run the command as root:

```
$ sudo python setup.py install
```

4.5 Copy Your Certs and Key File to PubSub Directory

For this example, we are going to have the cert and key files in the same directory as our Python script. In this case, we want to move the file to:

```
$ aws-iot-device-sdk-python/samples/basicPubSub
```

Copy the following files:

```
xxxxxxxxxx-certificate.pem.crt
xxxxxxxxxx-private.pem.key
root-CA.pem
```

to the above directory:

```
$ cp [cert location] aws-iot-device-sdk-python/samples/basicPubSub
$ cp [priv key location] aws-iot-device-sdk-python/samples/basicPubSub
$ cp [root CA location] aws-iot-device-sdk-python/samples/basicPubSub
```

Note: your filenames may be different than the above. What you need is your root certificate, device certificate, and private key files.

In addition to this, you will need the AWS IoT MQTT Host, which was provided in the guide linked in Section 4.1.

4.6 Start PPP Connection

Bring down your Ethernet or Wifi connection, and start the PPP connection according to this guide:

http://nimbelink.com/Documentation/Skywire/4G_LTE_Cat_M1/30273_NL-SW-LTE-SVZM21_PPP-on-CATM1_RPi.pdf

4.7 Run Example Script

Now that you have your endpoint, as well as your root CA certificate, device certificate, and private key in the same directory as the Python script, run the script according to the following syntax:

```
$ python basicPubSub.py -e [endpoint] -r [rootCAFilePath] -c
[certFilePath] -k [privateKeyFilePath]
```

Note: This description is available in the basicPubSub.py file.

So, for this example, we run:

```
$ python basicPubSub.py -e 1234567890abcd.iot.us-west-2.amazonaws.com
-r VeriSign-Class\ 3-Public-Primary-Certification-Authority-G5.pem -c
1234567890-certificate.pem.crt -k 1234567890-private.pem.key
```

Once you run this script, you should see something similar to below:

```
2017-04-18 13:39:24,121 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Load CAFile from:
VeriSign-Class 3-Public-Primary-Certification-Authority-G5.pem
2017-04-18 13:39:24,121 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Load Key from:
3640a6fd11-private.pem.key
2017-04-18 13:39:24,122 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Load Cert from:
3640a6fd11-certificate.pem.crt
2017-04-18 13:39:24,122 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Custom setting for
backoff timing: baseReconnectTime = 1 sec
2017-04-18 13:39:24,122 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Custom setting for
backoff timing: maximumReconnectTime = 32 sec
2017-04-18 13:39:24,123 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Custom setting for
backoff timing: minimumConnectTime = 20 sec
2017-04-18 13:39:24,123 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Custom setting for
publish queueing: queueSize = -1
2017-04-18 13:39:24,123 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Custom setting for
publish queueing: dropBehavior = Drop Newest
2017-04-18 13:39:24,124 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Custom setting for
draining interval: 0.5 sec
2017-04-18 13:39:24,124 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Set maximum
connect/disconnect timeout to be 10 second.
2017-04-18 13:39:24,124 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Set maximum MQTT
operation timeout to be 5 second
2017-04-18 13:39:24,125 - AWSIoTPythonSDK.core.protocol.mqttCore - INFO - Connection type: TLSv1.2
Mutual Authentication
2017-04-18 13:39:24,765 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Connect result code 0
2017-04-18 13:39:24,772 - AWSIoTPythonSDK.core.protocol.mqttCore - INFO - Connected to AWS IoT.
2017-04-18 13:39:24,772 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Connect time
consumption: 110.0ms.
2017-04-18 13:39:24,773 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Started a subscribe
request 1
2017-04-18 13:39:24,873 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - _resubscribeCount: -1
2017-04-18 13:39:24,874 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Subscribe request 1
sent.
2017-04-18 13:39:24,875 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Subscribe request 1
succeeded. Time consumption: 100.0ms.
```

```
2017-04-18 13:39:24,875 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Recover subscribe
context for the next request: subscribeSent: False
2017-04-18 13:39:26,878 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Try to put a publish
request 2 in the TCP stack.
2017-04-18 13:39:26,879 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Publish request 2
succeeded.
Received a new message:
New Message 0
from topic:
sdk/test/Python
--------------


2017-04-18 13:39:27,882 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Try to put a publish
request 3 in the TCP stack.
2017-04-18 13:39:27,883 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Publish request 3
succeeded.
Received a new message:
New Message 1
from topic:
sdk/test/Python
--------------


2017-04-18 13:39:28,885 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Try to put a publish
request 4 in the TCP stack.
2017-04-18 13:39:28,886 - AWSIoTPythonSDK.core.protocol.mqttCore - DEBUG - Publish request 4
succeeded.
Received a new message:
New Message 2
from topic:
sdk/test/Python
--------------
```

5. Next Steps

Amazon has multiple AWS IoT SDKs available for download, depending on your application and coding preference. You can view them all here:

http://docs.aws.amazon.com/iot/latest/developerguide/iot-sdks.html

Combined with NimbeLink's Skywire family of embedded cellular modems, you can develop your connected IoT device with ease.

6. Troubleshooting

Depending on your Skywire data connection, you may need to increase the timeout amount to allow enough time to work. For example, if you run into errors in Section 2, you can set the timeout by editing the file:

and modifying:

```
connectParams.mqttCommandTimeout_ms = 2000;
connectParams.tlsHandshakeTimeout_ms = 5000;
```

to larger values, such as `10000`.

Save the file, run the `make` command outlined in Amazon's guide, and try again.