



Agora | Getting started with AWS on Agora

3 Dec 2021 | v1.0.0

www.embeddedplanet.com



qualified
device
internet of things

1. Introduction

Check out the [AWS device catalog listing for Agora here!](https://devices.amazonaws.com/detail/a3G0h0000088JGgEAM/Embedded-Planet-Agora)

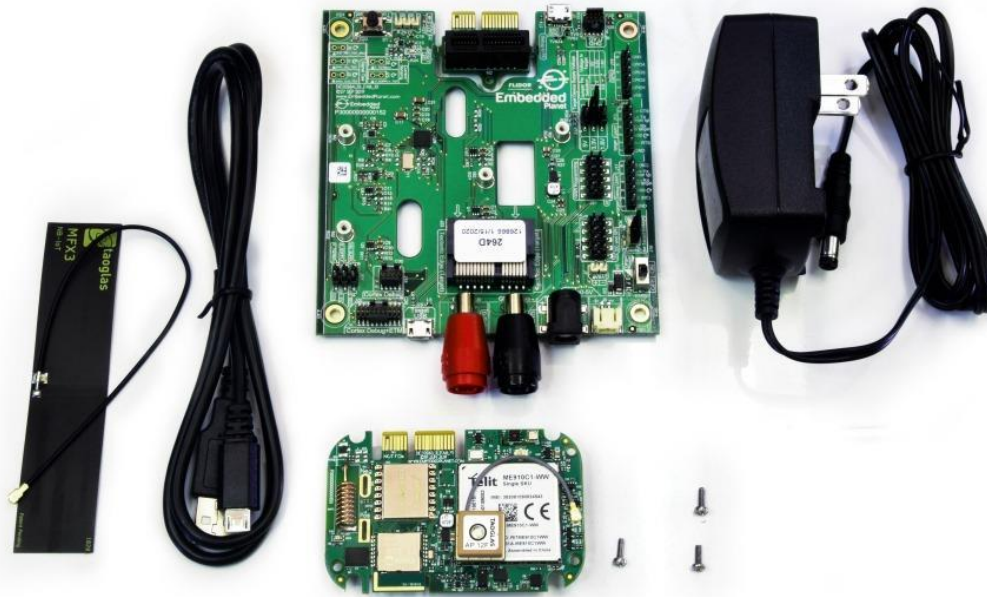
<https://devices.amazonaws.com/detail/a3G0h0000088JGgEAM/Embedded-Planet-Agora>

Our Agora platform is a production-ready solution that provides an excellent starting point for any cloud-connected design. This page will go over the necessary steps to get started using the Agora platform with AWS IoT.

If you run into any issues while trying to follow this guide please [contact our support team!](https://www.embeddedplanet.com/contact-us/)

<https://www.embeddedplanet.com/contact-us/>

2. Required Hardware



To follow along with this guide you just need an [Agora product development kit](https://shop.embeddedplanet.com/products/flidor-development-kit).
<https://shop.embeddedplanet.com/products/flidor-development-kit>

This kit includes an Agora module fully-loaded with all optional sensors and connectivity options, as well as a Flidor debug board, an AC adapter, and a compatible cellular antenna.

3. REQUIRED SOFTWARE AND SUPPORTED TOOLCHAINS

a. Build Tools Installation

The Agora product development kit supports software development using ARM's IoT RTOS, [Mbed-OS](https://os.mbed.com/mbed-os/).
<https://os.mbed.com/mbed-os/>

Follow the [Mbed-OS guide here](https://os.mbed.com/docs/mbed-os/v6.2/quick-start/build-with-mbed-cli.html) to install the Mbed CLI build tools and build your first application. Due to the complexity of this project, we **do not** recommend using the online IDE/compiler to follow this guide.
<https://os.mbed.com/docs/mbed-os/v6.2/quick-start/build-with-mbed-cli.html>

b. Supported IDEs and Toolchains

Mbed-OS build tools support both the commercially available ARMCC6 compiler and the free GNU ARM toolchains.

Mbed-OS supports development and debugging using a variety of popular IDEs, including:

- Mbed Studio
- VSCode
- Eclipse
- Code::Blocks

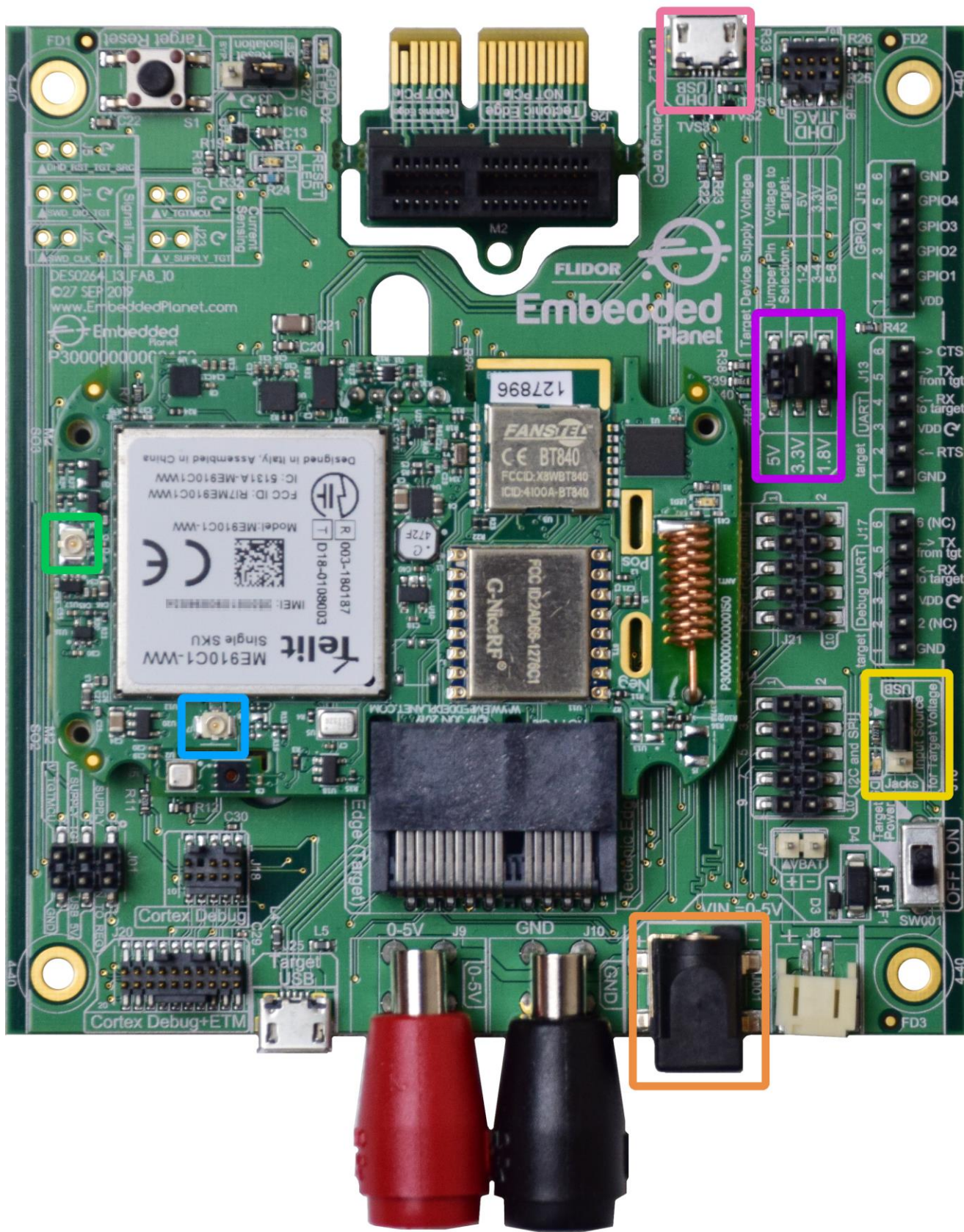
and many more. You can learn more about supported IDEs and toolchains, as well as how to export Mbed projects and set up debugging from Mbed's [official documentation here](https://os.mbed.com/docs/mbed-os/v6.2/build-tools/third-party-build-tools.html).

<https://os.mbed.com/docs/mbed-os/v6.2/build-tools/third-party-build-tools.html>

4. Other Software

Additionally, you will need a serial terminal program to view debug output from your Agora board. The Flidor development board (included in the Agora product development kit) has an on-board UART-to-USB converter so no other adapters are required. This guide will show using minicom on Ubuntu. Follow your serial terminal software's documentation on how to set up a similar connection.

5. Hardware Setup



Before running the demo, you must verify the following are properly set up on your Agora product development kit: the cellular antenna, the power supply, and the power supply selection jumper.

For this project you must have the included cellular antenna plugged in to the appropriate U.FL connector, **J7**, on the Agora board. In the above image, the cellular antenna connector is highlighted with a **light blue box**. The other U.FL connector, J8, highlighted with a **light green box**, is for the GPS antenna. **Make sure the Agora board is being powered by the included DC power supply.** The power supply included with the Flidor development kit can be plugged into the DC jack highlighted with an **orange box**. Attempting to power the Agora board for cellular applications using USB may cause cellular connectivity issues. A typical USB port cannot provide enough current in some scenarios.

You must also locate jumper J16 on the Flidor development board, highlighted in the image above with a **yellow box**, it is adjacent to the power switch. Make sure to move the jumper to the position marked "Jacks", as shown in the image above, so the Agora module is powered from the DC jack input rather than the USB input. The Target Device Supply Voltage jumper setting (J12), highlighted with a **violet box** in the image above, should be set to 3.3V as shown above.

Plug the Flidor development board into your computer's USB port, making sure to use the micro USB Type-B port labeled "DHD USB", highlighted with a **pink box** in the image above. The Flidor board should show up as a small removable drive and as a serial port.

Your hardware is now set up and ready to connect to AWS!

6. SETTING UP AWS

This section will go over the necessary steps to get the required AWS services set up on your account. If you do not already have an AWS account, you can create one for free using the [free tier subscription](https://docs.aws.amazon.com/iot/latest/developerguide/setting-up.html).
<https://docs.aws.amazon.com/iot/latest/developerguide/setting-up.html>

Follow the steps outlined in the [guide to Create AWS IoT resources found here](https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs-first-thing.html).
<https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs-first-thing.html>

You should follow the steps in that guide to:

- Create an IoT policy
- Create your first thing
- Generate a device certificate and keys

Once you get to the section titled "Configure your device" you may stop and return to this tutorial.

Your AWS instance should be ready to go!

7. SETTING UP THE MBED-OS APPLICATION

The example Mbed-OS firmware application for this tutorial [can be cloned from GitHub here](https://github.com/EmbeddedPlanet/mbed-os-example-aws).
<https://github.com/EmbeddedPlanet/mbed-os-example-aws>

Simply clone or download a copy of that repository and open up a command line in that project. In the command line, execute the following to download all required dependencies:

```
mbed deploy && mbed config root .
```

Set EP_AGORA as your build target: `mbd target EP_AGORA`. Also make sure to set the toolchain you are using, eg: `mbd toolchain GCC_ARM`.

Once these steps are complete, it's time to integrate your previously-downloaded device credentials into the code and configure the application.

8. INTEGRATING CREDENTIALS

The example includes a python script to automate converting the credentials you downloaded from AWS into C-compatible arrays/strings. First, create a new folder in the project to store your credential files, eg: `mkdir aws-credentials`.

Then, you can run the script to automatically generate the necessary code from the credentials:

```
python aws-cert-converter.py aws-credentials
```

For more details on how to use the convert script, simply pass in the `-h` flag to print the help documentation.

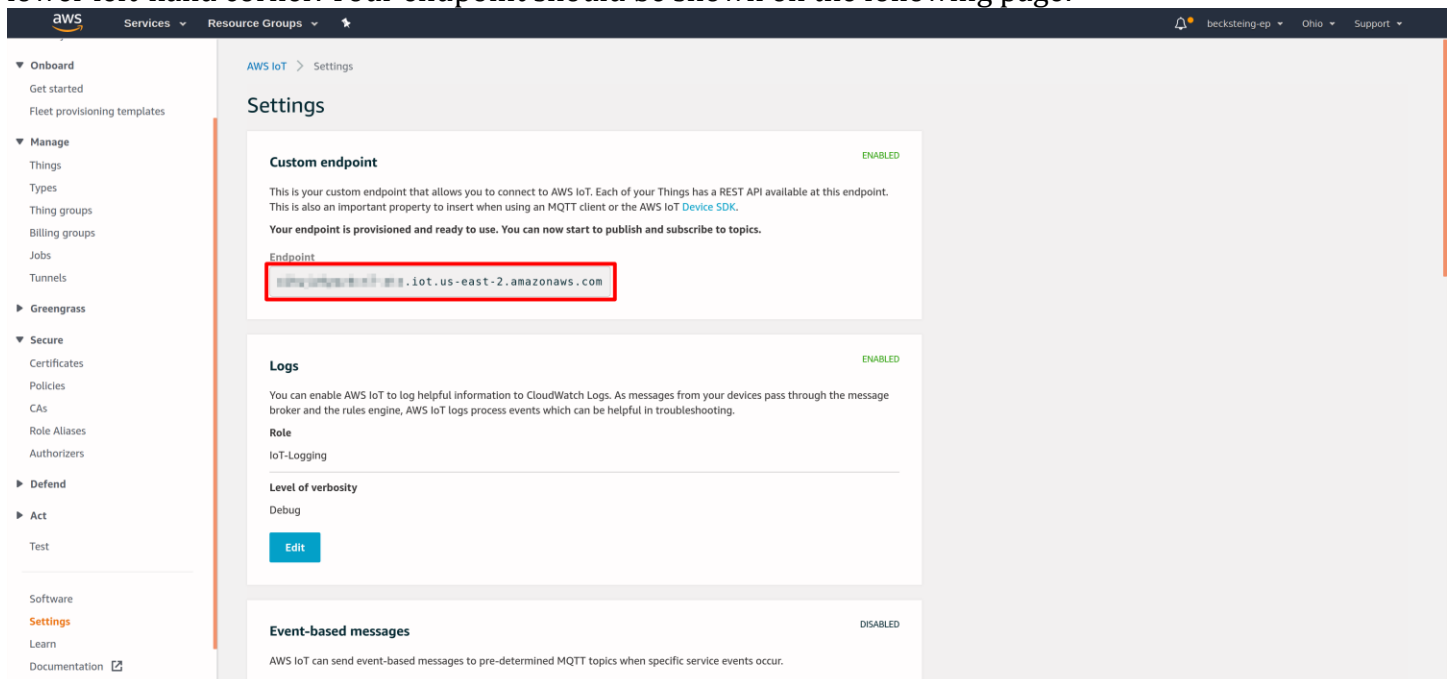
The above command will read your credential files and place them into a C header file: `aws_credentials.h`. Your credentials are now integrated into the code!

9. CONFIGURING THE APPLICATION

Before building the application, you must configure a few things including the URL of your AWS IoT instance and an MQTT topic to publish/subscribe to. To do this, open up the existing `mbd_app.json` file from the example project in a text editor.

Under the config section you will see three configuration parameters, `aws-endpoint`, `aws-mqtt-topic`, and `aws-client-identifier`.

The value field for the `aws-endpoint` parameter must be replaced with your AWS IoT Instance's specific "Custom endpoint" URL. To find this, open the AWS IoT Console Webpage and click on "Settings" in the lower left-hand corner. Your endpoint should be shown on the following page:



Replace the value of the `aws-endpoint` field with your AWS custom endpoint URL. Make sure to retain the leading `"\"` and trailing `\"`!

The next parameter, `aws-mqtt-topic`, can be left as-is or changed to a specific topic of your choice.

The last parameter, `aws-client-identifier`, can be set to something that will uniquely identify your thing, such as the name you gave it when creating the thing in AWS.

Once these are configured you can build the example with the following command: `mbed compile`.

10. RUNNING THE EXAMPLE

Once the build finishes, simply program the Agora board with the compiled binary. You can accomplish this by dragging the `mbed-os-example-aws.bin` file from the `BUILD/EP_AGORA/GCC_ARM` folder to the DAPLINK removable drive that shows up when you plug in the Flidor development board.

After programming, open your preferred serial terminal of choice to view the debug UART output. The Flidor development board also shows up as a USB serial port. The example application's debug output baud rate is set to 115200 by default.

You should see output similar to the following:

```
[INFO][Main]: Connecting to the network...
[INFO][CELL]: New CellularContext (0x20011AF0)
[INFO][CELL]: CellularContext connect
[INFO][CELL]: Start connecting (timeout 1000 ms)
[INFO][CELL]: Modem power ON (timeout 1000 ms)
[INFO][CELL]: Modem manufacturer: Telit

[INFO][CELL]: Modem model: ME910C1-WW

[INFO][CELL]: Modem revision: M0B.800004

[INFO][CELL]: Modem ready
[INFO][CELL]: Setup SIM (timeout 1000 ms)
[INFO][CELL]: SIM is ready
[INFO][CELL]: RSSI -63 dBm
[INFO][CELL]: Roaming 0 Registered 1
[INFO][CELL]: Registering network => Attaching network
[INFO][CELL]: RSSI -63 dBm
[INFO][CELL]: Attaching network (timeout 60000 ms)
(...)
[INFO][CELL]: Found PDP context 1
[INFO][CELL]: CellularContext PPP connect
[INFO][CELL]: CellularContext IP 10.246.6.214
[INFO][Main]: MAC:
[INFO][Main]: Connection Success
[INFO][INIT][01 Jan 1970 00:00] SDK successfully initialized.
[INFO][MQTT][01 Jan 1970 00:00] MQTT library successfully initialized.
[INFO][TLSW]: Starting TLS handshake with <snipped>.iot.us-east-2.amazonaws.com
[INFO][TLSW]: TLS connection to <snipped>.iot.us-east-2.amazonaws.com established
[INFO][TLSW]: Certificate verification passed
[INFO][MQTT][01 Jan 1970 00:00] Establishing new MQTT connection.
```

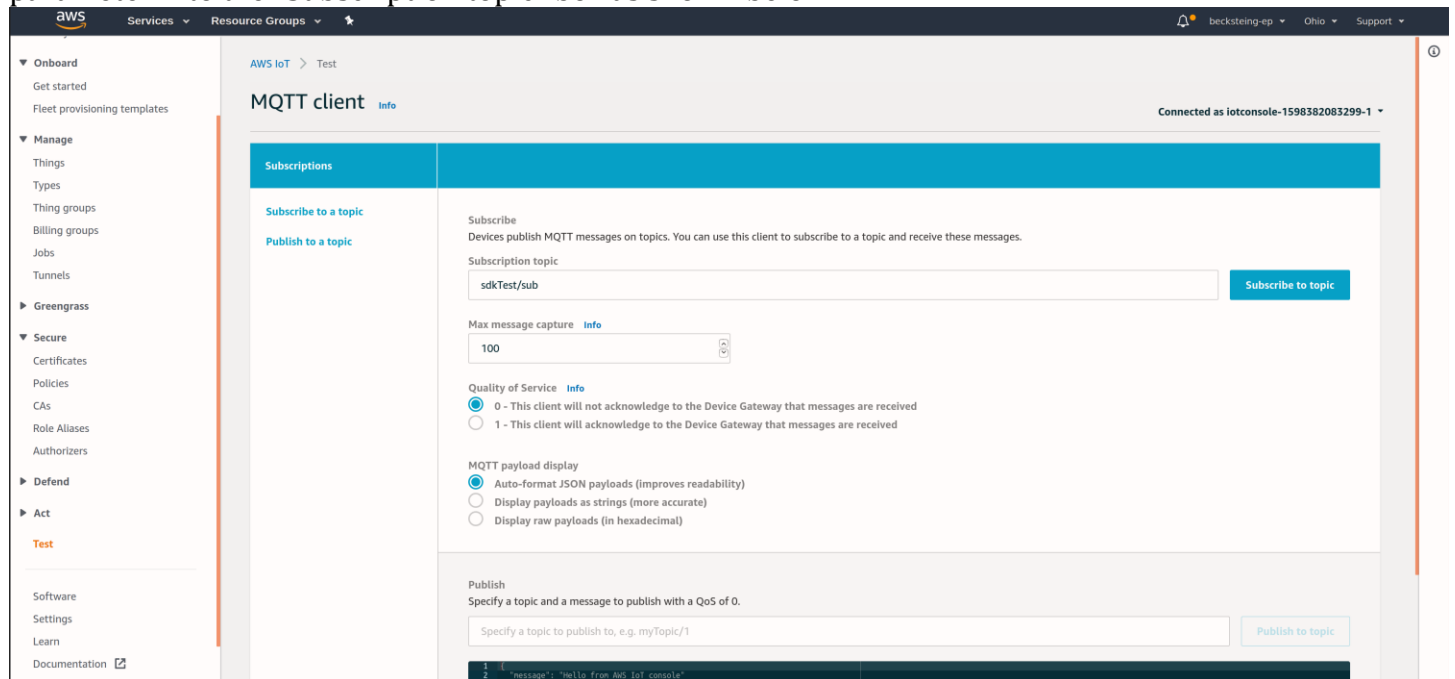


```
[INFO ][MQTT][01 Jan 1970 00:00] Anonymous metrics (SDK language, SDK version) will be provided to
AWS IoT. Recompile with AWS_IOT_MQTT_ENABLE_METRICS set t.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, CONNECT operation 0x20020B28) Waiting
for operation completion.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, CONNECT operation 0x20020B28) Wait
complete with result SUCCESS.
[INFO ][MQTT][01 Jan 1970 00:00] New MQTT connection 0x200106A0 established.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) SUBSCRIBE operation scheduled.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, SUBSCRIBE operation 0x20021698)
Waiting for operation completion.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, SUBSCRIBE operation 0x20021698) Wait
complete with result SUCCESS.
[INFO ][Main]: sending warning message: Warning: Only 10 second(s) left to say your name !
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) MQTT PUBLISH operation queued.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x20021768) Waiting
for operation completion.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x20021768) Wait
complete with result SUCCESS.
[INFO ][TASKPOOL][01 Jan 1970 00:00] Growing a Task pool with a new worker thread...
[INFO ][Main]: sending warning message: Warning: Only 9 second(s) left to say your name !
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) MQTT PUBLISH operation queued.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x20020B28) Waiting
for operation completion.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x20020B28) Wait
complete with result SUCCESS.
(...)
[INFO ][Main]: sending warning message: Warning: Only 2 second(s) left to say your name !
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) MQTT PUBLISH operation queued.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x200211D8) Waiting
for operation completion.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x200211D8) Wait
complete with result SUCCESS.
[INFO ][Main]: sending warning message: Warning: Only 1 second(s) left to say your name !
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) MQTT PUBLISH operation queued.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x200211D8) Waiting
for operation completion.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x200211D8) Wait
complete with result SUCCESS.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) Disconnecting connection.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, DISCONNECT operation 0x20020B28)
Waiting for operation completion.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, DISCONNECT operation 0x20020B28) Wait
complete with result SUCCESS.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) Connection disconnected.
[INFO ][TLW]: Closing TLS
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) Network connection closed.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) Network connection destroyed.
[INFO ][MQTT][01 Jan 1970 00:00] MQTT library cleanup done.
[INFO ][INIT][01 Jan 1970 00:00] SDK cleanup done.
[INFO ][Main]: Done
```

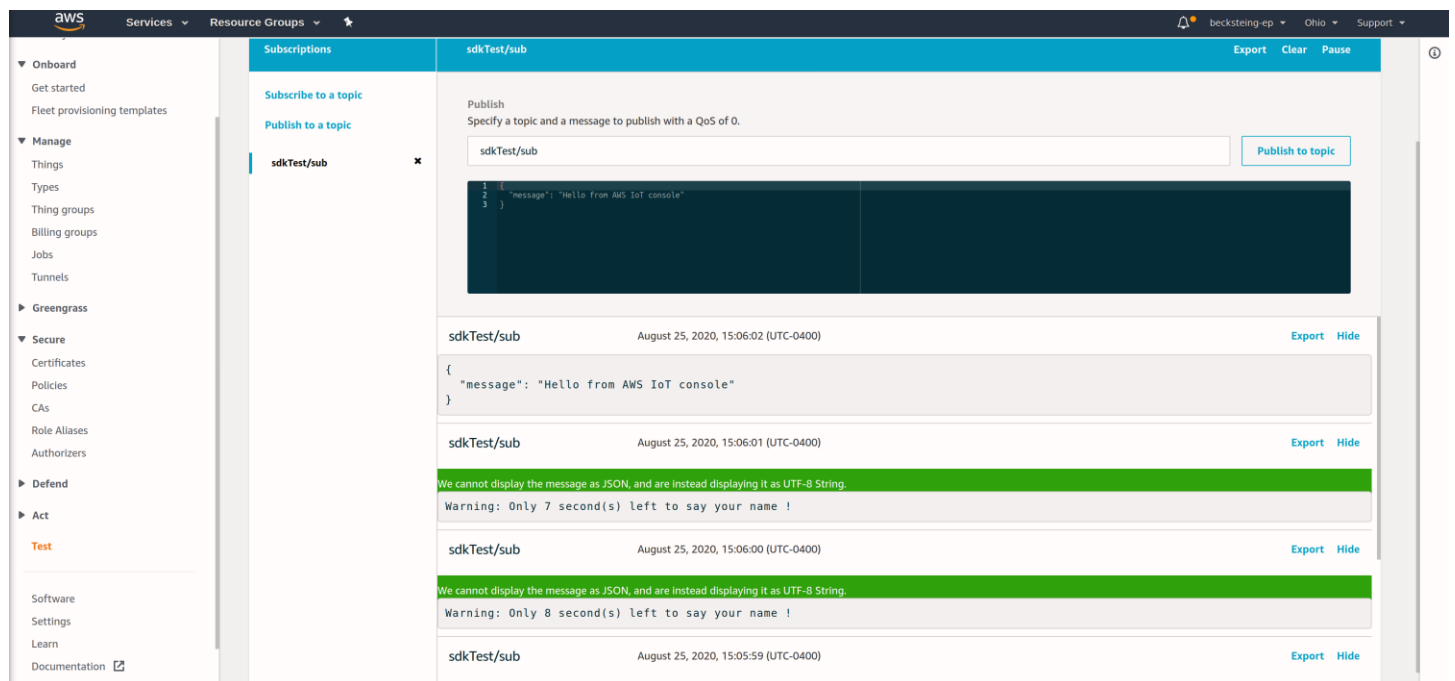
If so, congratulations! The Agora AWS example is running successfully!

If you have any trouble look through the below Troubleshooting section to see if it helps solve your problem. You can also reach out to Embedded Planet's support team using [our website's Contact Us form](#).

You can see the published messages on the cloud by using the AWS IoT Console. In the lower-left corner of the IoT console page, click "Test". This will open a diagnostics MQTT client where you can subscribe to the topic you configured in the last section. Copy the value field from the `aws-mqtt-topic` configuration parameter into the "Subscription topic" box as shown below:



Then, reset your Agora target (Flidor has an integrated target reset button), to restart the example. In the AWS IoT MQTT Client you should start to see messages from the target as shown below:



If you click the "Publish to topic" button during the 10-second timeout period of the example, your device should print the message to the serial terminal!

The serial output should look similar to below:

```
[INFO][Main]: sending warning message: Warning: Only 7 second(s) left to say your name !
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) MQTT PUBLISH operation queued.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x200211D8) Waiting
for operation completion.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, PUBLISH operation 0x200211D8) Wait
complete with result SUCCESS.
[INFO][Main]: Hello {
  "message": "Hello from AWS IoT console"
} !
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) Disconnecting connection.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, DISCONNECT operation 0x20021698)
Waiting for operation completion.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0, DISCONNECT operation 0x20021698) Wait
complete with result SUCCESS.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) Connection disconnected.
[INFO][TLFW]: Closing TLS
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) Network connection closed.
[INFO ][MQTT][01 Jan 1970 00:00] (MQTT connection 0x200228D0) Network connection destroyed.
[INFO ][MQTT][01 Jan 1970 00:00] MQTT library cleanup done.
[INFO ][INIT][01 Jan 1970 00:00] SDK cleanup done.
[INFO][Main]: Done
```

11. TROUBLESHOOTING

A few troubleshooting tips to solve common problems in this example:

- If the example project fails to compile, make sure the mbed-os version is appropriate. At time of writing, you can ensure the mbed-os version is compatible by executing the following: `cd mbed-os && git checkout mbed-os-6.2.1`
- If the example project fails to connect, you should verify you are in an area with good cellular signal strength. You may want to try relocating the Agora board and retry the example. You should also make sure the cellular antenna is properly oriented. For best performance, the cellular antenna should be perpendicular relative to the ground.
- Make sure your Agora board has a SIM card in the on-board socket.
- Make sure your cellular APN is configured as appropriate for your carrier. You can change this parameter by modifying the `nsapi.default-cellular-apn` value in the example application's `mbed_app.json` configuration file.
- If you are having issues connecting to AWS, authenticating, or publishing/subscribing over MQTT, you can view cloud-side error logs by enabling AWS Cloudwatch Logging for your AWS IoT instance. [Follow this guide to view AWS IoT logs in this way.](#)