

Agora | Getting started with Azure on Agora

3 Dec 2021 | v1.0.0



Microsoft Azure
IoT Hub



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 Azure IoT Hub.

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

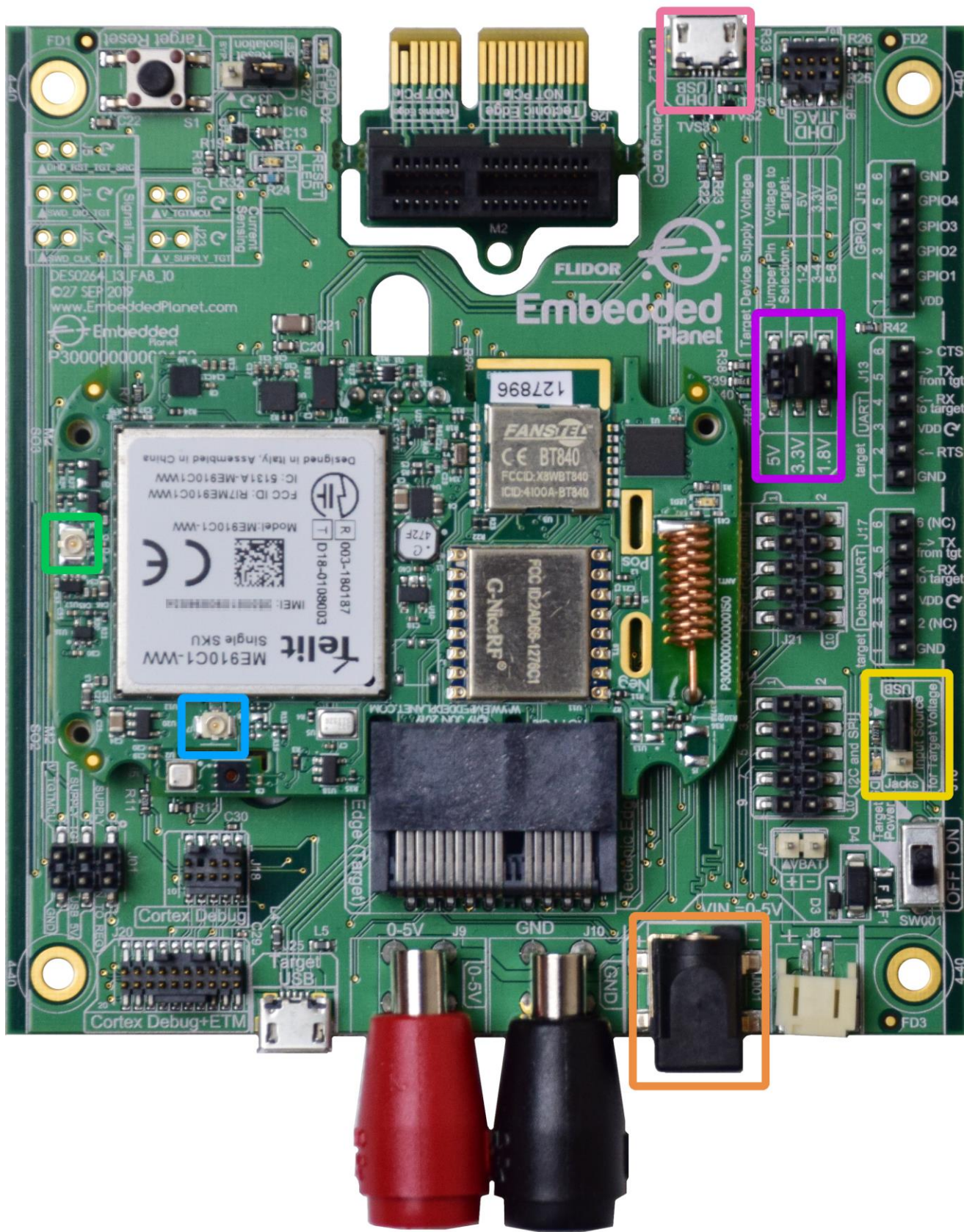
and many more. You can learn more about supported IDEs and toolchains, as well as how to export Mbed projects and set up debugging from Mbed's [official documentation here](https://os.mbed.com/docs/mbed-os/v6.2/build-tools/third-party-build-tools.html).

<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 AZURE

Follow Azure IoT Hub's official documentation to

1. Create a new hub on the Azure portal ([documentation](#)). You will need a **Standard tier** hub to enable cloud-to-device messages for this example, and a free option is available in this tier.
<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-create-through-portal#create-an-iot-hub>
<https://azure.microsoft.com/en-gb/pricing/details/iot-hub/>
2. Register a new device to the hub you have created ([documentation](#)). Make a copy of the "Primary Connection String" of the device. You may stop and return to this tutorial when you reach the section titled "Message Routing for an IoT Hub."
<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-create-through-portal#register-a-new-device-in-the-iot-hub>

When you are finished creating your new device, return to the "IoT Devices" list in your IoT Hub portal. Refresh the list if your new device doesn't show up at first. Click the name of your new device to open a page with device credentials and details you will need to connect later. On this page, copy the "Primary Connection String" and paste it in a temporary file for use later:

Microsoft Azure

Search resources, services, and docs (G+)

Home > epiotHub >

myNewDeviceId

epiotHub

Save

Message to Device

Direct Method

Add Module Identity

Device Twin

Manage keys

Refresh

Device ID

myNewDeviceId

Primary Key

.....

Secondary Key

.....

Primary Connection String

.....

Secondary Connection String

.....

Enable connection to IoT Hub

☒ Enable
 ☐ Disable

Parent device

No parent device

Module Identities

Configurations

MODULE ID	CONNECTION STATE	CONNECTION STATE LAST UPDATED (UTC)	LAST ACTIVITY TIME (UTC)
There are no module identities for this device.			

Note: The Primary Connection String contains sensitive information that authenticates your new device to your IoT Hub instance. It should be treated like a password!
Now the IoT Hub is ready for use in this example!

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: `mbed target EP_AGORA`. Also make sure to set the toolchain you are using,
eg: `mbed toolchain GCC_ARM`.

Once these steps are complete, it's time to integrate your device's Primary Connection String.

8. INTEGRATING CREDENTIALS

Integrating your device's Primary Connection String into this example application is very simple, there's no coding required! Simply open the application configuration file, `mbed_app.json`, in a text editor and update the `value` parameter under the `iothub_connection_string` configuration. There's no manual formatting necessary, simply copy the device's Primary Connection String from the previous steps and paste it in place of the `"HostName=...;DeviceId=...;SharedAccessKey=..."`, make sure to retain the leading `"\"` and trailing `\""`!

Your credentials are now integrated into the code!

9. CONFIGURING THE APPLICATION

Before building the application, you must configure the APN of your cellular carrier along with any other cellular credentials your SIM may require, such as a PIN code, username, and/or password.

Under the `EP_AGORA` parameters in `mbed_app.json`, replace the value of `nsapi.default-cellular-apn` with one that is appropriate for your SIM and carrier. Make sure to retain the leading `"\"` and trailing `\""`!

The parameters you may need to modify in your `mbed_app.json` are highlighted in the screenshot below:

```
{
  "config": {
    "iothub_connection_string": {
      "help": "Azure IoT Hub connection string",
      "value": "\"HostName=...;DeviceId=...;SharedAccessKey=...\""
    },
    "iothub_client_trace": {
      "help": "Enable IoT Hub Client tracing",
      "value": false
    }
  },
  "target_overrides": {
    "L475VG_IOT01A": {
      "platform.stdio-convert-newlines": true,
      "platform.stdio-baud-rate": 115200
    },
    "DISCO_L475VG_IOT01A": {
      "target.components-add": ["wifl_ism43362"],
      "target.network-default-interface-type": "WIFI",
      "nsapi.default-wifi-security": "WPA_WPA2",
      "nsapi.default-wifi-ssid": "\"SSID\"",
      "nsapi.default-wifi-password": "\"PASSWORD\""
    },
    "EP_AGORA": {
      "nsapi.default-cellular-apn": "\"APN\"",
      "nsapi.default-cellular-sim-pin": null,
      "nsapi.default-cellular-username": null,
      "nsapi.default-cellular-password": null,
      "platform.stdio-buffered-serial": true,
      "platform.stdio-flush-at-exit": true,
      "drivers.uart-serial-rxbuf-size": 1024,
      "drivers.uart-serial-txbuf-size": 1024,
      "lwip.ipv4-enabled": true,
      "lwip.ipv6-enabled": true,
      "lwip.ppp-enabled": true,
      "lwip.tcp-enabled": true,
      "lwip.ethernet-enabled": false,
      "lwip.mem-size": 22000,
      "lwip.tcpip-thread-stacksize": 2000,
      "nsapi.dns-response-wait-time": 30000,
      "lwip.use-mbed-trace": true,
      "lwip.debug-enabled": false,
      "target.features_remove": ["CRYPTOCELL310"],
      "target.macos_remove": ["MBEDTLS_CONFIG_HW_SUPPORT"],
      "target.macos_add": ["NRF_RNG_ENABLED=1", "RNG_ENABLED=1", "NRF_QUEUE_ENABLED=1"]
    }
  }
}
```

In most cases, you will only need to modify the `iothub_connection_string` and `nsapi.default-cellular-apn` parameters.

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-for-azure.hex` 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: Connecting to the network
Info: Connection success, MAC:
Info: Getting time from the NTP server
Info: Time: Tue Oct 20 16:57:14 2020

Info: RTC reports Tue Oct 20 16:57:14 2020

Info: Starting the Demo
Info: Initializing IoT Hub client
Info: Sending: "10 messages left to send, or until we receive a reply"
Info: Sending: "9 messages left to send, or until we receive a reply"
-> 16:57:18 CONNECT | VER: 4 | KEEPALIVE: 240 | FLAGS: 192 | USERNAME: epiothub0
<- 16:57:18 CONNACK | SESSION_PRESENT: false | RETURN_CODE: 0x0
Info: Connected to IoT Hub
-> 16:57:18 SUBSCRIBE | PACKET_ID: 2 | TOPIC_NAME: devices/myNewDeviceId/message1
-> 16:57:19 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 3
-> 16:57:19 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2
<- 16:57:19 SUBACK | PACKET_ID: 2 | RETURN_CODE: 1
Info: Sending: "8 messages left to send, or until we receive a reply"
-> 16:57:19 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2
<- 16:57:19 PUBACK | PACKET_ID: 3
Info: Message sent successfully
<- 16:57:19 PUBACK | PACKET_ID: 4
Info: Message sent successfully
<- 16:57:19 PUBACK | PACKET_ID: 5
Info: Message sent successfully
Info: Sending: "7 messages left to send, or until we receive a reply"
-> 16:57:20 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2
<- 16:57:20 PUBACK | PACKET_ID: 6
Info: Message sent successfully
Info: Sending: "6 messages left to send, or until we receive a reply"
-> 16:57:21 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2
<- 16:57:21 PUBACK | PACKET_ID: 7
Info: Message sent successfully
Info: Sending: "5 messages left to send, or until we receive a reply"
-> 16:57:22 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2
<- 16:57:22 PUBACK | PACKET_ID: 8
Info: Message sent successfully
Info: Sending: "4 messages left to send, or until we receive a reply"
-> 16:57:23 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2
<- 16:57:23 PUBACK | PACKET_ID: 9
```

Info: Message sent successfully

Info: Sending: "3 messages left to send, or until we receive a reply"

-> 16:57:24 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2

<- 16:57:24 PUBACK | PACKET_ID: 10

Info: Message sent successfully

Info: Sending: "2 messages left to send, or until we receive a reply"

-> 16:57:25 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2

<- 16:57:25 PUBACK | PACKET_ID: 11

Info: Message sent successfully

Info: Sending: "1 messages left to send, or until we receive a reply"

-> 16:57:26 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2

<- 16:57:26 PUBACK | PACKET_ID: 12

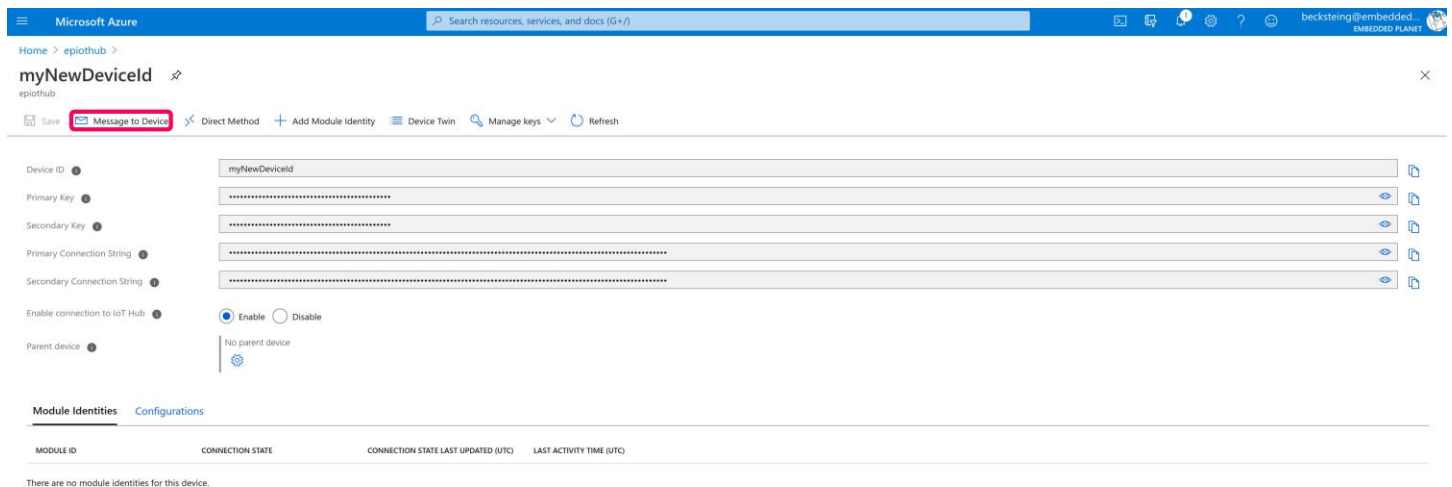
Info: Message sent successfully

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

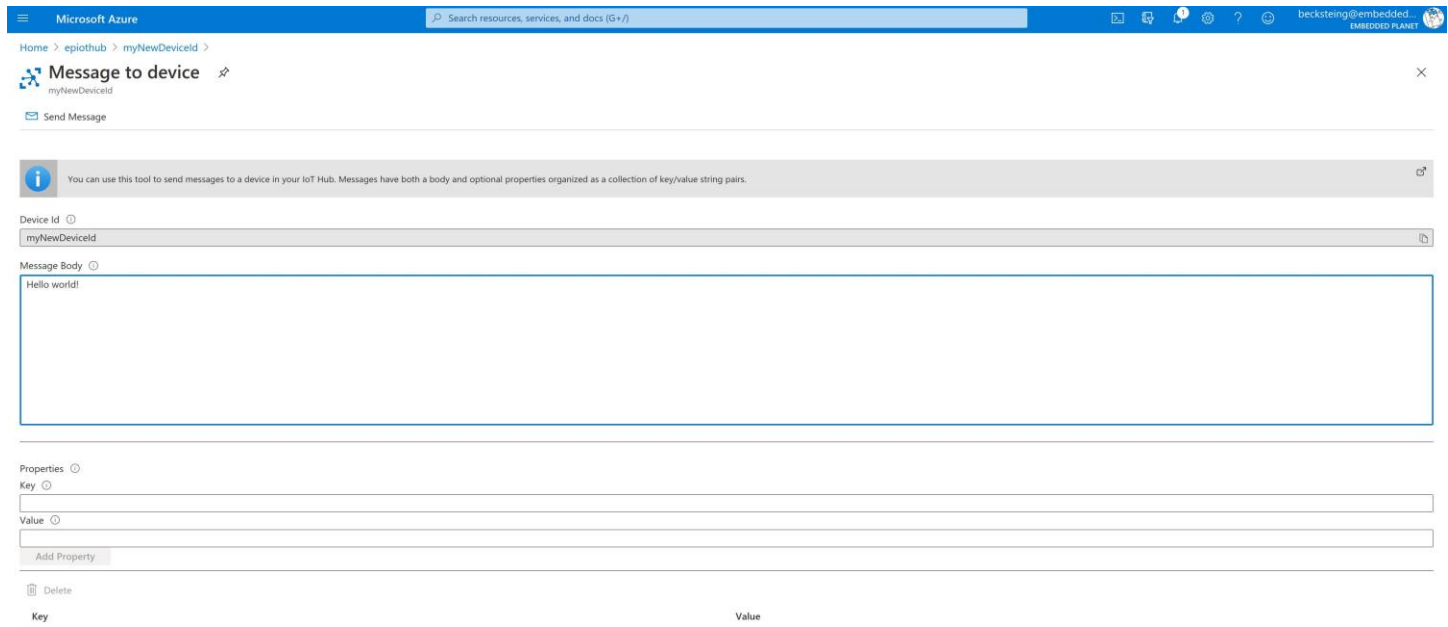
If you have any trouble look through the below Troubleshooting section to see if it helps solve your problem. You can also reach out to Embedded Planet's support team using [our website's Contact Us form](#).

11. Sending messages from cloud to device

This demo also allows you to test sending messages from Azure IoT Hub to the device. To try this out, first navigate to the following page in your Azure portal. Click the "IoT devices" tab in the left pane as before, navigate to the device you're running the example with and click into its details page. Once there, you can send a message to the device from the portal by clicking the "Message to Device" option as shown in the screenshot below:



Once the next page loads, you can type your message in "Message Body" text box:



Before sending the message, reset the Agora microcontroller (the Flidor development board has a convenient on-board reset button) and wait for the output to show that the device has connected to Azure IoT Hub successfully. While the board is counting down you can send the message and the output should show successful receipt of your message as below:

Info: Connecting to the network

Info: Connection success, MAC:

Info: Getting time from the NTP server

Info: Time: Tue Oct 20 18:50:56 2020

Info: RTC reports Tue Oct 20 18:50:56 2020

Info: Starting the Demo

Info: Initializing IoT Hub client

Info: Sending: "10 messages left to send, or until we receive a reply"

Info: Sending: "9 messages left to send, or until we receive a reply"

-> 18:51:00 CONNECT | VER: 4 | KEEPALIVE: 240 | FLAGS: 192 | USERNAME: epiotHub0

<- 18:51:00 CONNACK | SESSION_PRESENT: true | RETURN_CODE: 0x0

Info: Connected to IoT Hub

-> 18:51:00 SUBSCRIBE | PACKET_ID: 2 | TOPIC_NAME: devices/myNewDeviceId/messag1

-> 18:51:01 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 3

-> 18:51:01 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2

Info: Sending: "8 messages left to send, or until we receive a reply"

-> 18:51:01 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | 2

<- 18:51:01 SUBACK | PACKET_ID: 2 | RETURN_CODE: 1

<- 18:51:01 PUBACK | PACKET_ID: 3

<- 18:51:01 PUBACK | PACKET_ID: 4

Info: Message sent successfully

Info: Message sent successfully

<- 18:51:01 PUBACK | PACKET_ID: 5

Info: Message sent successfully

```
<- 18:51:02 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_MOST_ONCE | T2  
Info: Message received from IoT Hub  
Info: Message body: Hello world!  
-> 18:51:02 DISCONNECT  
Info: The demo has ended
```

12. 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.](#)