COM



Contents

[Step 1: Pre-requisites](#_heading=h.gjdgxs)

[Step 2: Set up Azure IoT Central](#_heading=h.x83i5pfahv7v)

[Step 3: Adding devices to your template](#_heading=h.3dy6vkm)

[Step 4: Building the connection string in node.js](#_heading=h.1t3h5sf)

[Step 5: Configure your IoT device for Azure](#_heading=h.krl2dbujhayd)

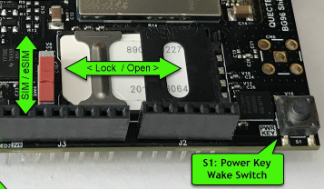
[Step 6: Viewing the data being transmitted and recieved](#_heading=h.2s8eyo1)

[Step 7: Adding widgets to your dashboard](#_heading=h.lnxbz9)

# Step 1: Pre-requisites

## Configure your IoT Hardware

Ensure the SIM switch is in the SIM position, and the SIM is inserted with the notch close to the switch.



Also, please ensure that the Rx/Tx slide switches are set as shown below (move switches away from the BG96 chip):



Connect the BG96 with sensor module to the L496 MCU so it looks like below:



## Sofware and services configuration

1. [Git](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git) - used to download the MBED Command Line during the setup your device.
2. [Mercurial](https://www.mercurial-scm.org/downloads) - a required Distributed Version Control command-line tool for MBED CLI
3. [Python 2.7](https://www.python.org/downloads/release/python-2713/) (already built into MacOS)
4. [GNU ARM Embedded Toolchain](https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads) - is a ready-to-use, open source suite of tools for C, C++ and Assembly languages targeting Arm processors. The processor used on our MCU board is a ARM processor.
5. [Azure Command-Line Tools](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest) - Microsoft's cross-platform command-line experience for managing Azure resources. It will be used in the Monitoring payloads sent to Azure section.

For window users, add Python to your user or systems PATH environment variable

**TIP** Python 2 is required for this tutorial, if you have both Python 2 and 3 installed you can substitute python in the command-line with py -2 for the remainder of this tutorial which will ensure Python 2 is being used\*

## Install PIP, the Python Package Installer

PIP is a command-line tool that installs Python packages, it is the standard for installing requirements for Python projects and we will need to use it to gather dependencies before we can compile the MBED-OS.

1. From the command-line run the following command to retrieve the PIP install script:

* curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

1. Run the following command to retrieve and install PIP:

* python get-pip.py

Verify PIP is installed correctly and ensure your Python setuptools package is up-to-date by running the following command:

* python -m pip install --upgrade setuptools
* If you encounter errors with the above command, try appending --user and re-run

That's all for PIP for now, we'll reference it again a bit later.

## MBED Command Line (mbed-cli)

The mbed-cli is hosted on github and built in Python, so we can download it using git and compile using Python, now that we have made sure both are installed on our computer.

From the command-line:

1. git clone https://github.com/ARMmbed/mbed-cli.git
2. cd mbed-cli
3. python setup.py install

Now you should be able to run the mbed command from your command-line, you may need to relaunch your terminal for it to work. In Windows you may need to instead run python -m mbed.

## Download the Avnet Azure IoT Client

Avnet has created a client for the TELUS IoT starter kit that, with a couple of configuration tweaks, is ready to compile and load onto your IoT board.

Get the client downloaded by running the following from the command-line, this will create a folder with loads of files, so be sure to run the command in a folder that works for you:

1. mbed import https://github.com/Avnet/azure-iot-mbed-client

Note: In Windows you may need to prepend that command with python -m

The import will take a while, and we can’t do too much more with the client until we get Azure up and running, so let’s jump over to Azure to get things rolling on that side.

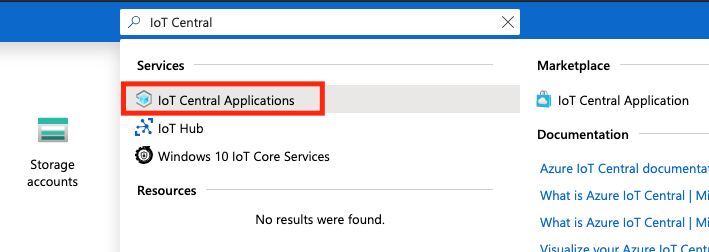
# Step 2: Set up Azure IoT Central

**Setting Up Your Azure Account**

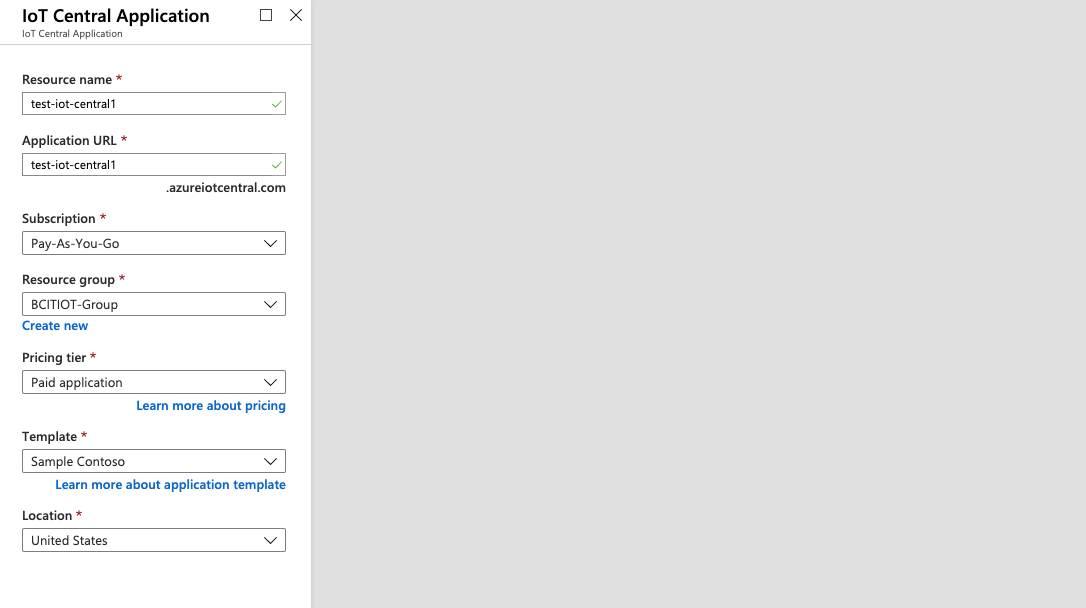
Microsoft’s Azure is incredibly useful cloud platform that has built-in support for IoT and allows for simple integration with several other services. If you don’t already have an Azure account you can sign up for a free trial which comes bundled with $250 of free credits: <https://azure.microsoft.com/en-ca/>

**Creating Your IoT Central**

Once you have your account created you can proceed to create a new IoT Central from your Azure search bar and search for “IoT Central Application



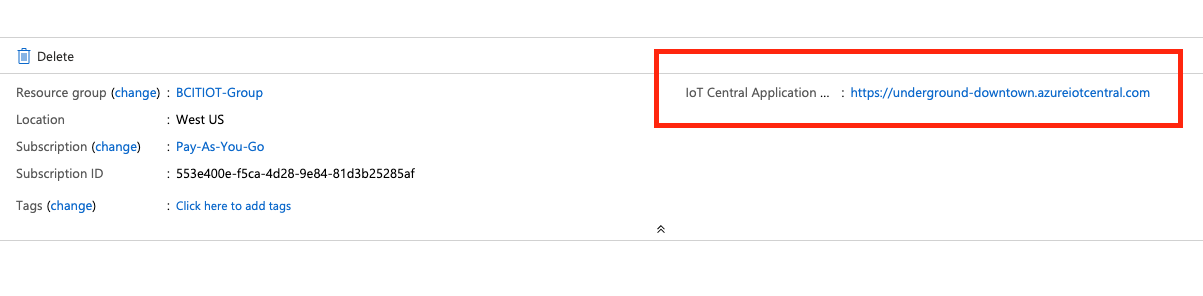
Select “Add” and create a new IoT Central Application, select Sample Contoso for template if you don’t have a template in mind.



Create your instance. This may take a couple of minutes.

Now our IoT Central is created! This will be our central location for all our IoT devices to connect and send whatever data we have configured them to relay, and gives us a single point to read and action on that data. Azure has security built-in, all communications between our IoT devices to Azure will be secured and visibility to that data is also protected.

Open your newly created IoT Central instance, then click on the url next to IoT Central Application URL

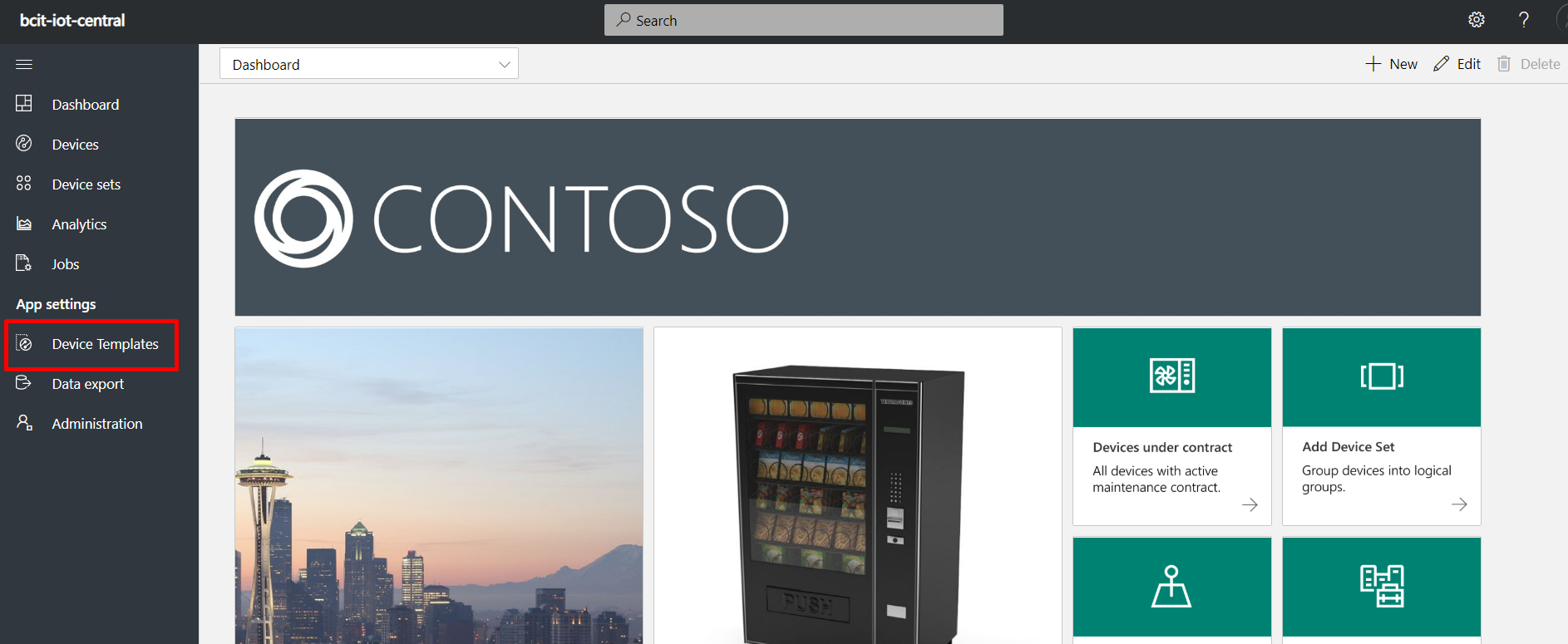


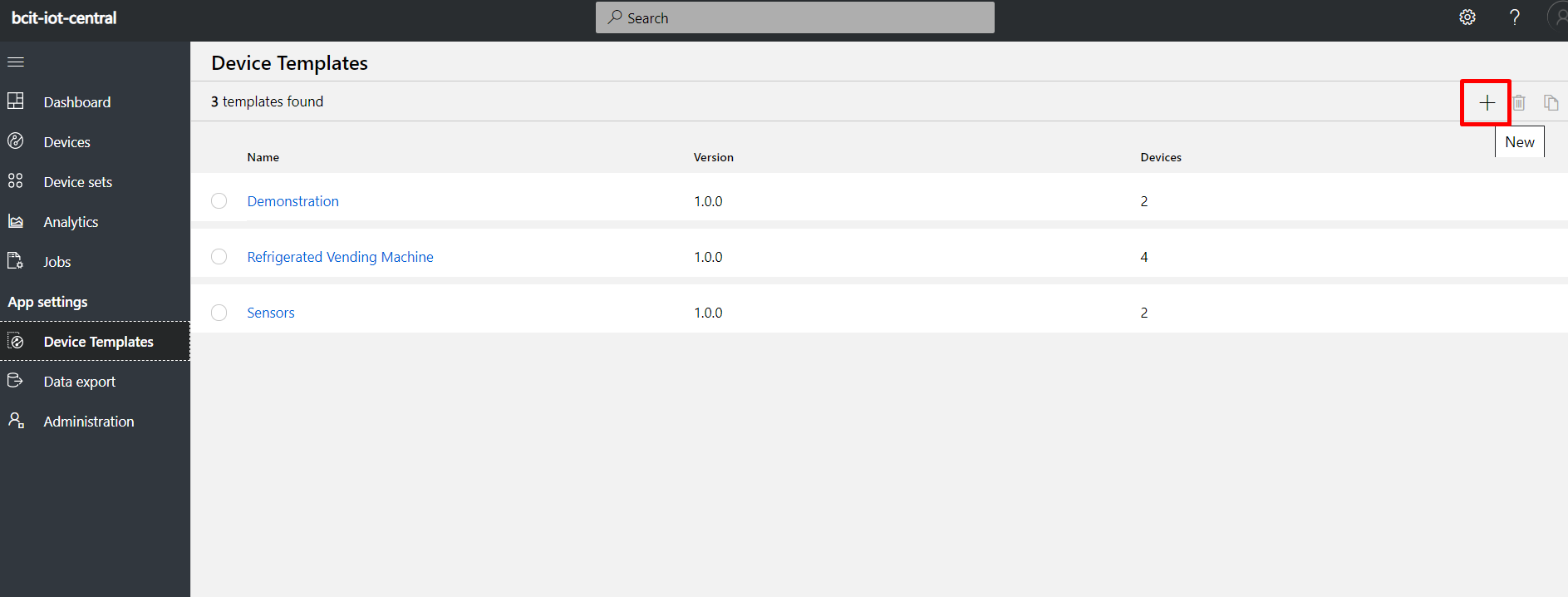
This will navigate you to your Central Application page.

## Create a template for your devices

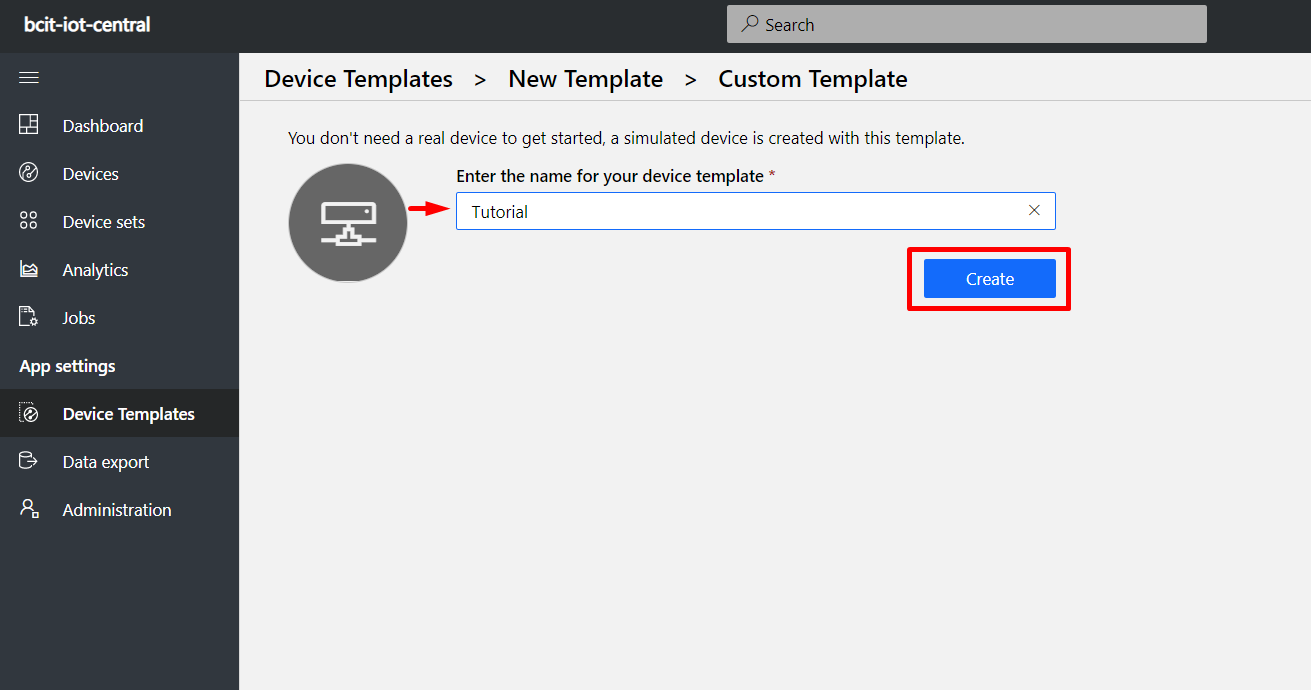
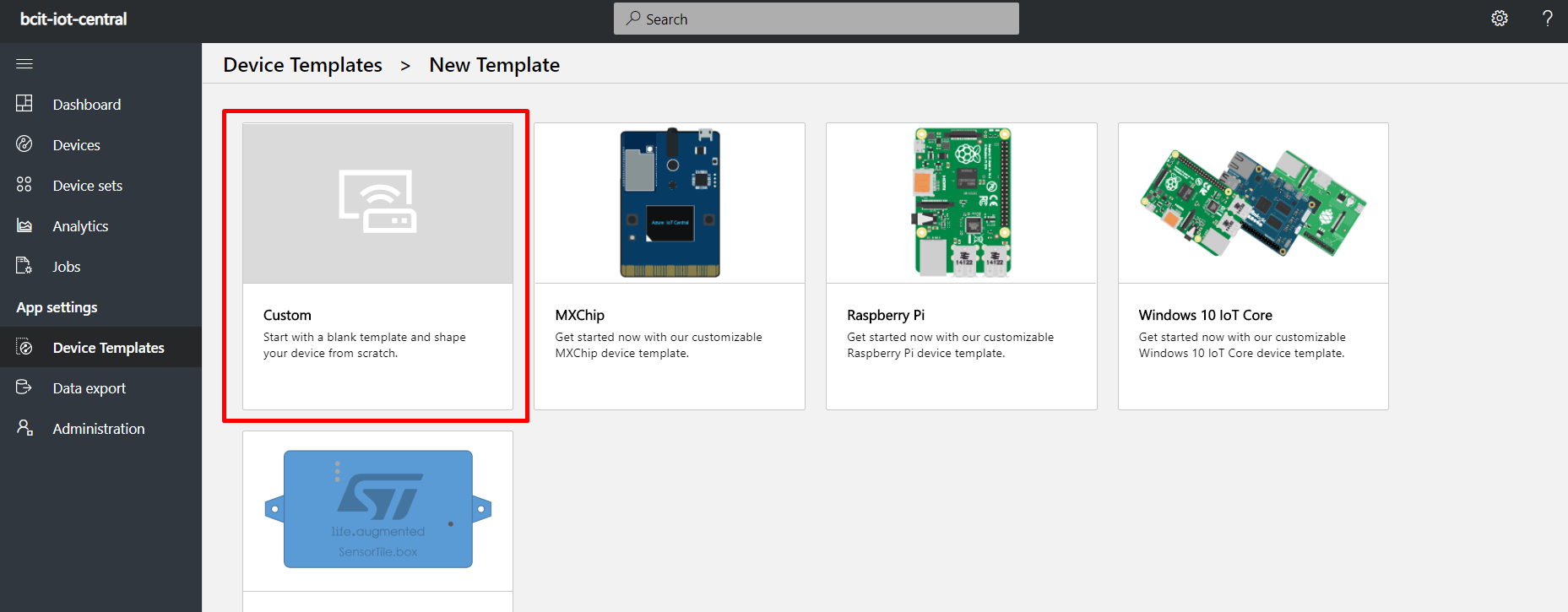
Here you will be able to create a template for all of your devices by adding sensor data formats.

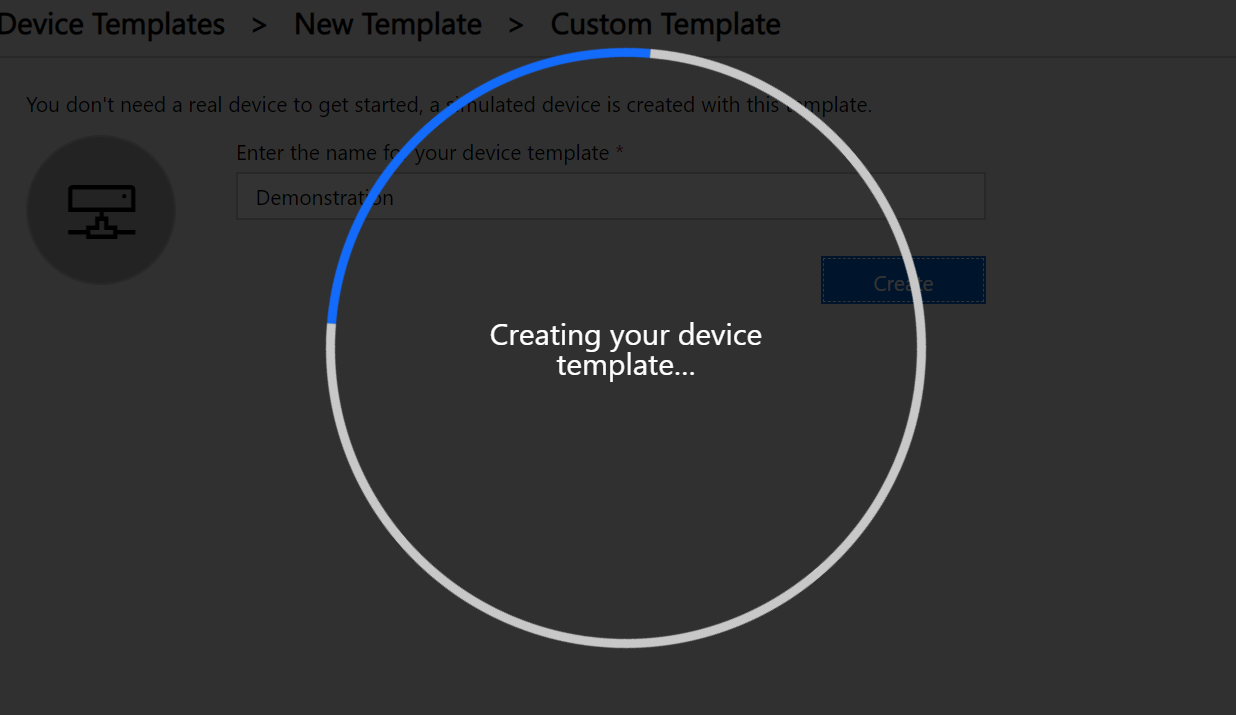
Select Device Template in the side menu



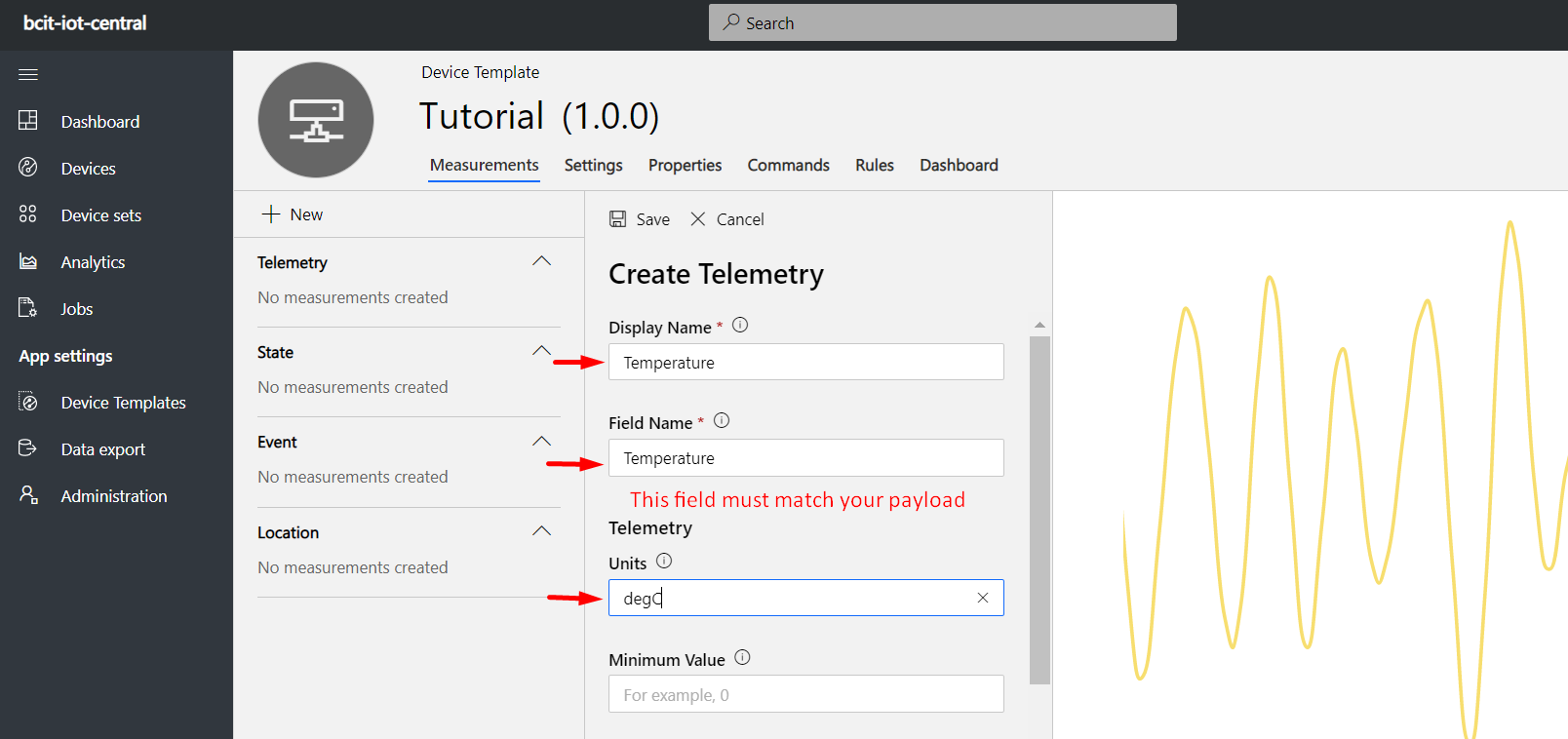
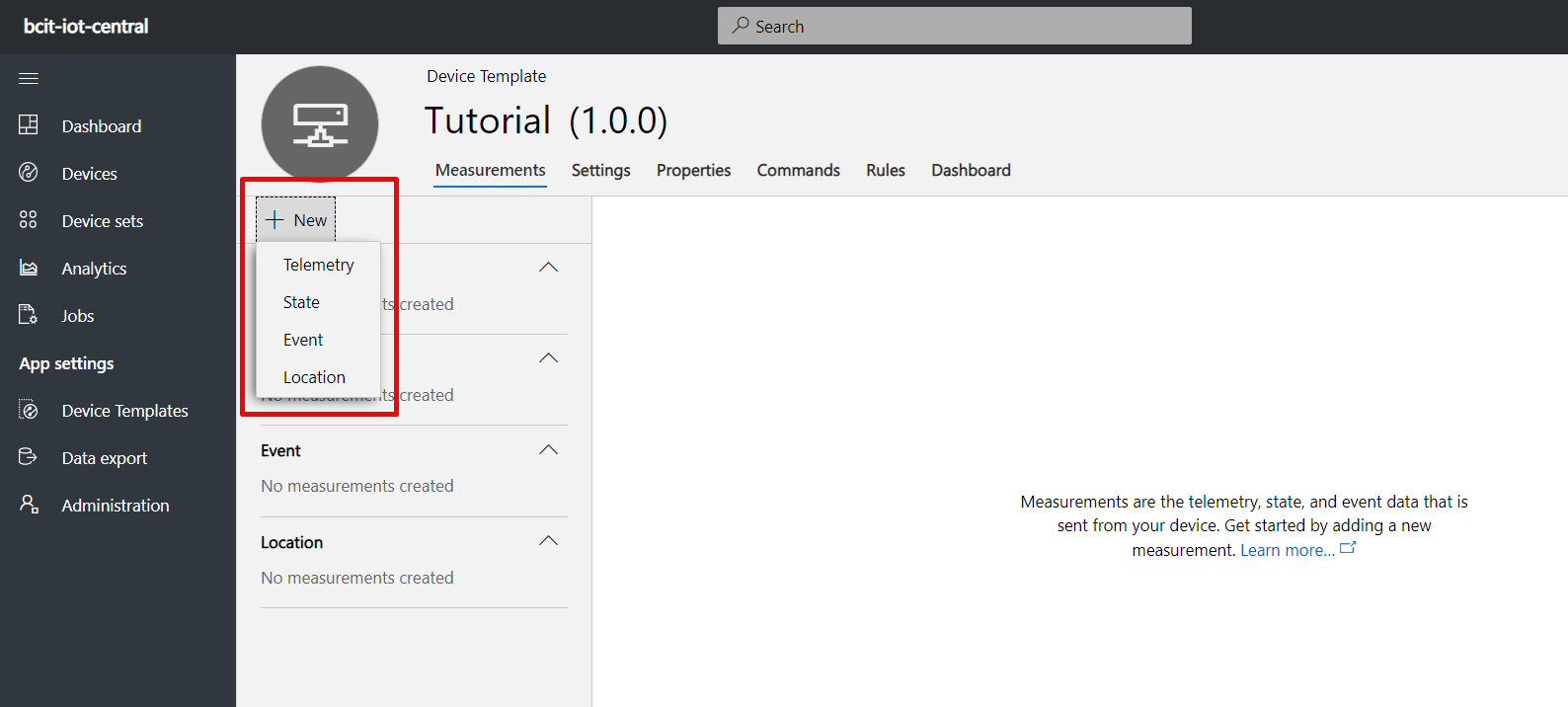


Select “Custom”





Add the sensor data that matches what your device will be transmitting to IoT Central.



Note: The field name must match the names in the code, for instance, the default will look like this.

"\"location\":\"%6.3f\"," \

"\"GPSTime\":\"%6.0f\","      \

"\"GPSDate\":\"%s\","         \

"\"Temperature\":\"%.02f\","  \

"\"Humidity\":\"%d\","        \

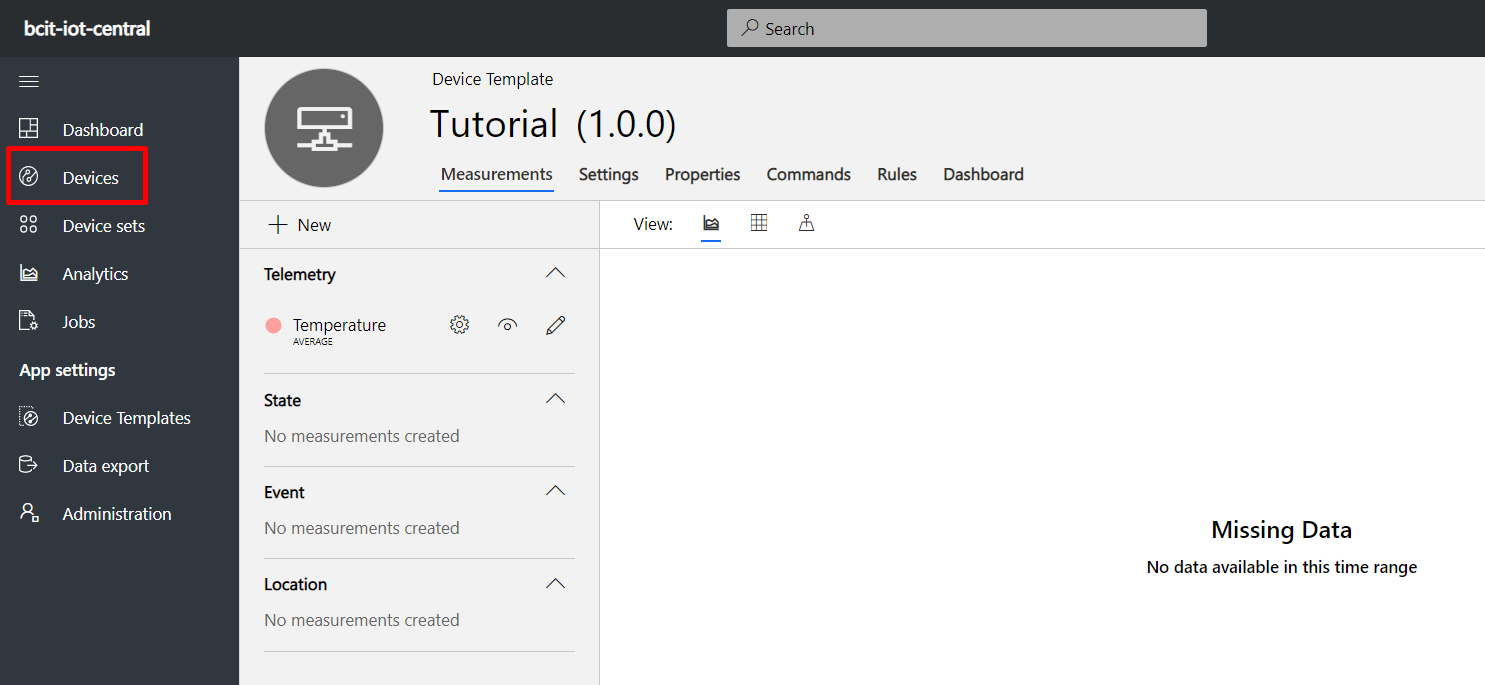
"\"Pressure\":\"%d\","        \

"\"Tilt\":\"%d\","            \

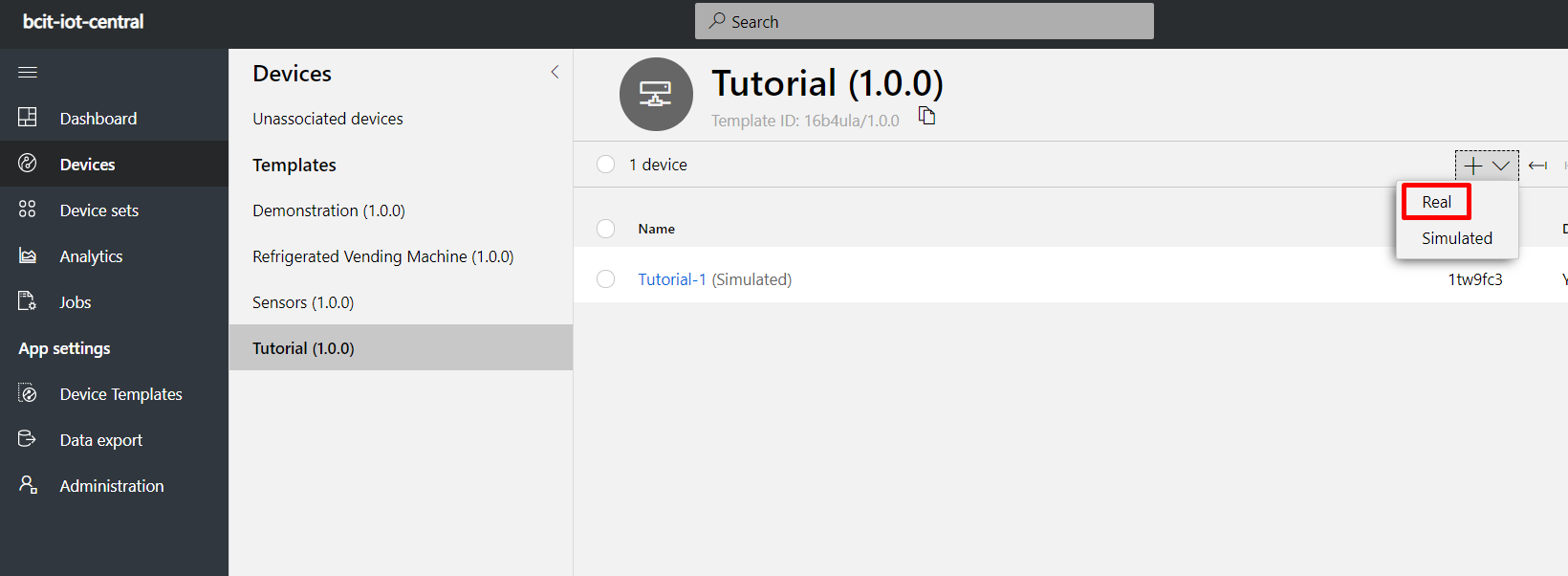
"\"ButtonPress\":\"%d\","     \

# Step 3: Adding devices to your template

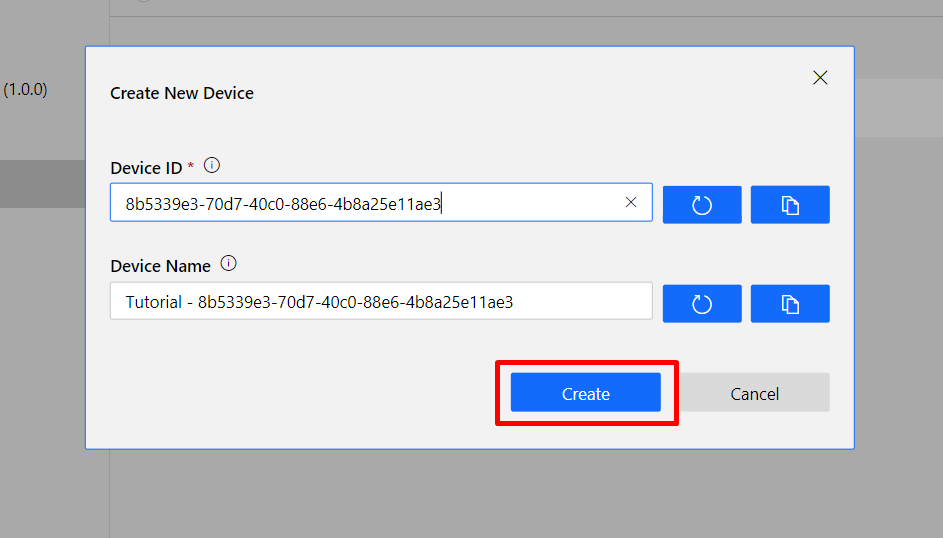
1. Select Devices on the side panel



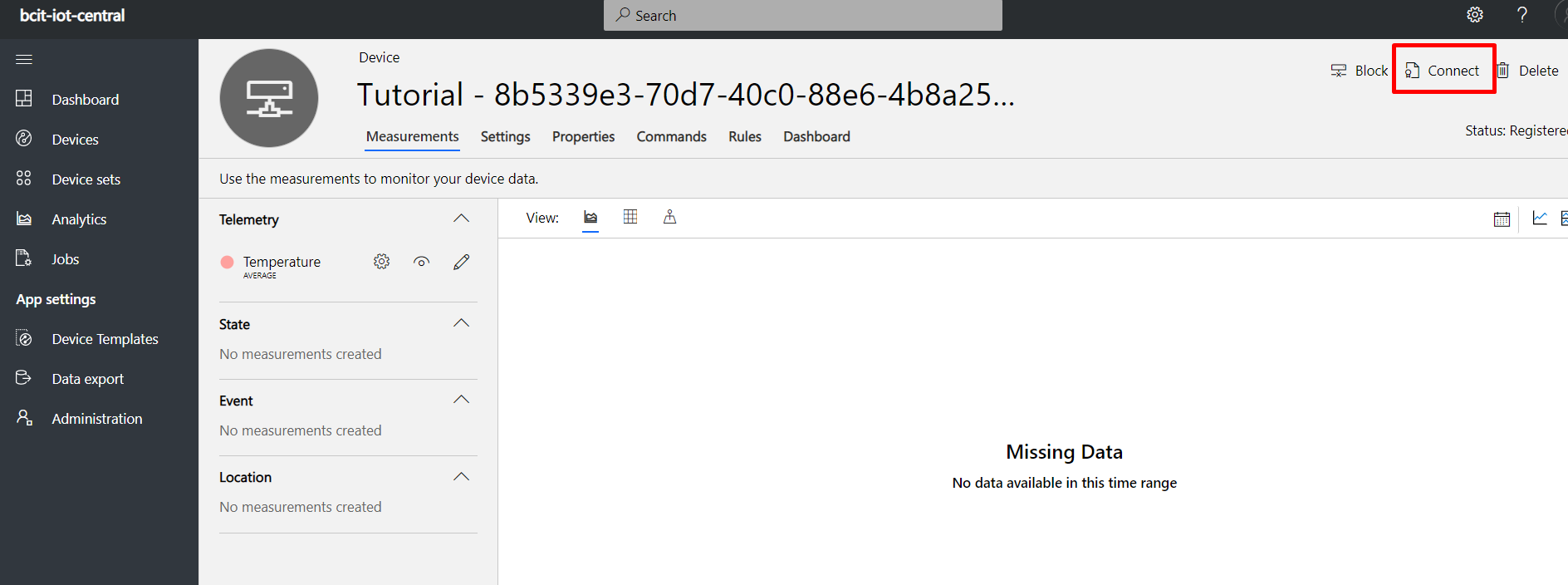
1. Select + button and select Real.



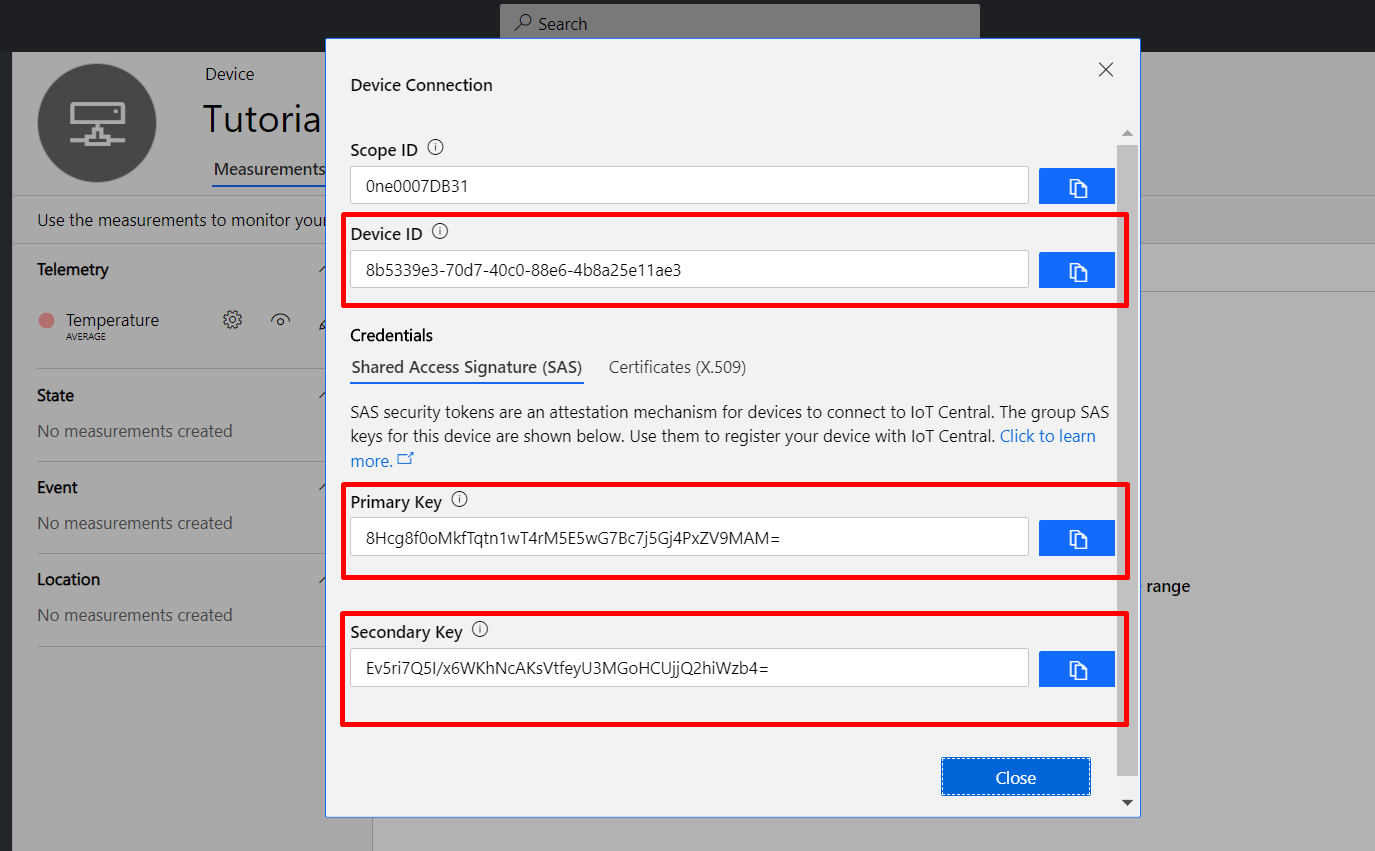
1. Create a new Device



1. Select “Connect” on the top right corner



6. Copy “Scope ID”, “Device ID” and “Primary Key” to a text file.

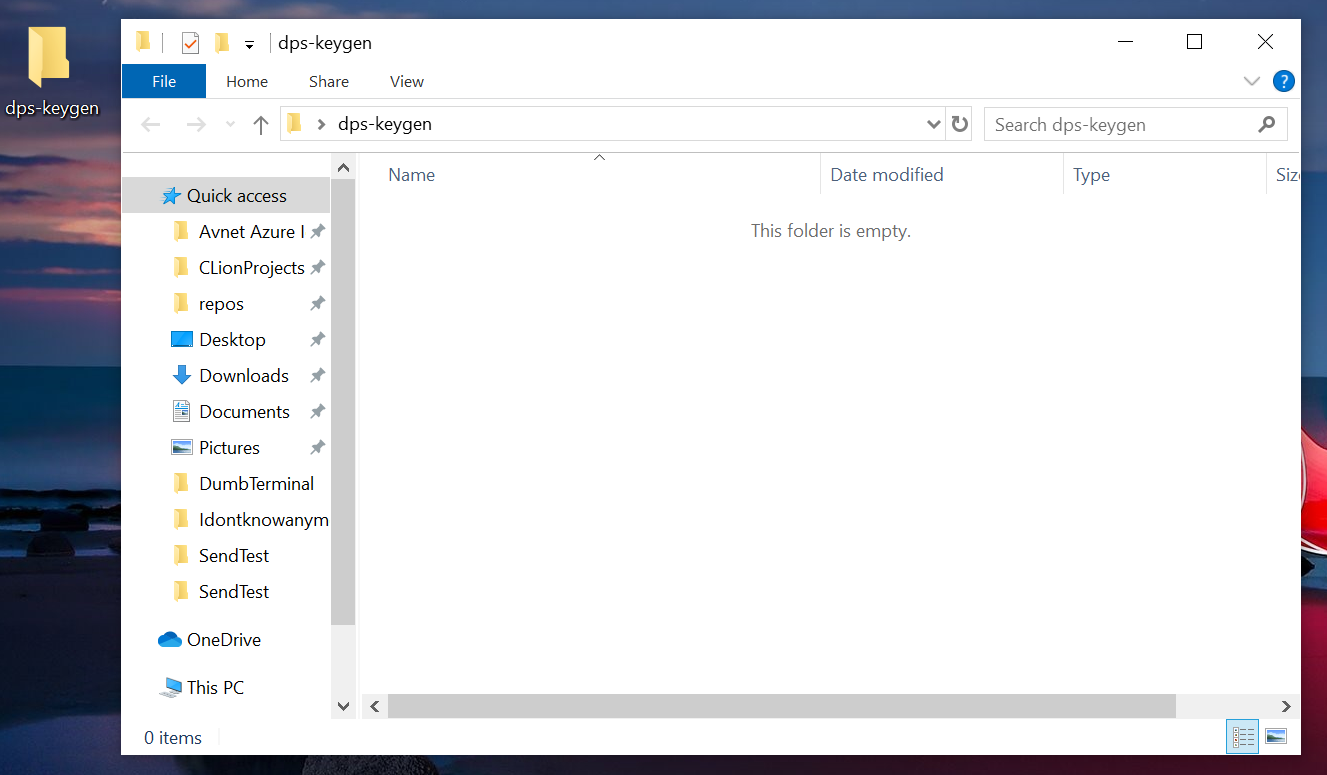


# Step 4: Building the connection string in node.js

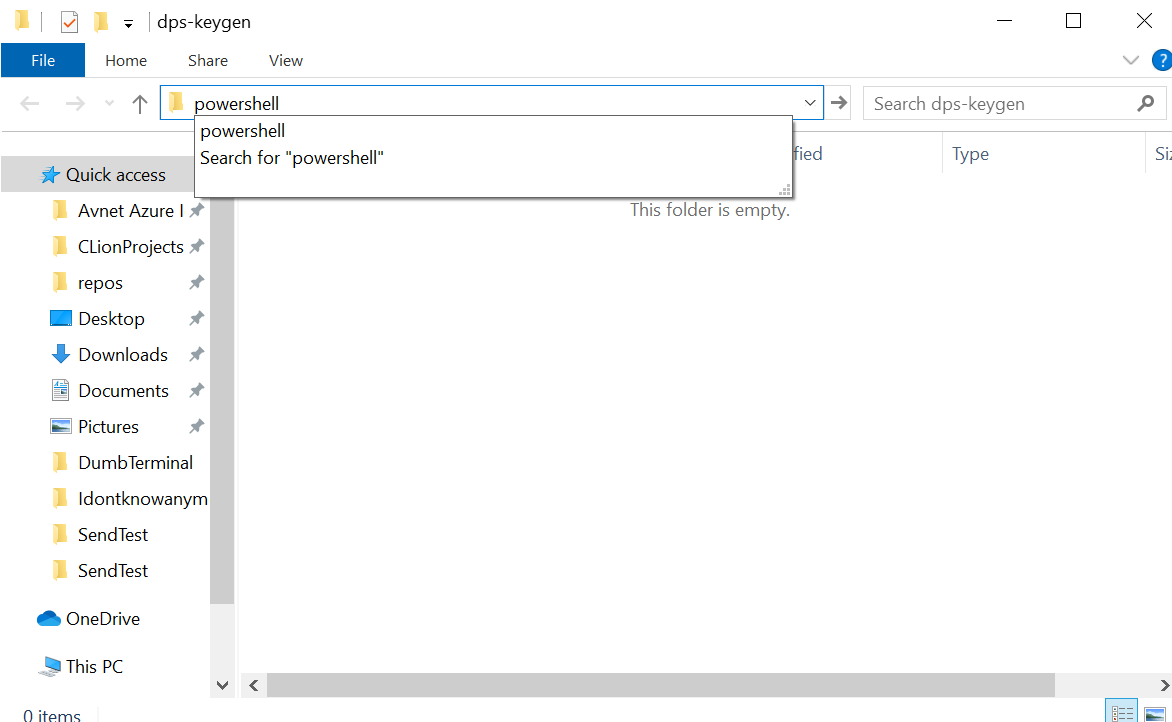
<https://github.com/Azure/dps-keygen>

This section will cover taking the information of the device created in the last step and fetch the connection string from Microsoft Azure.

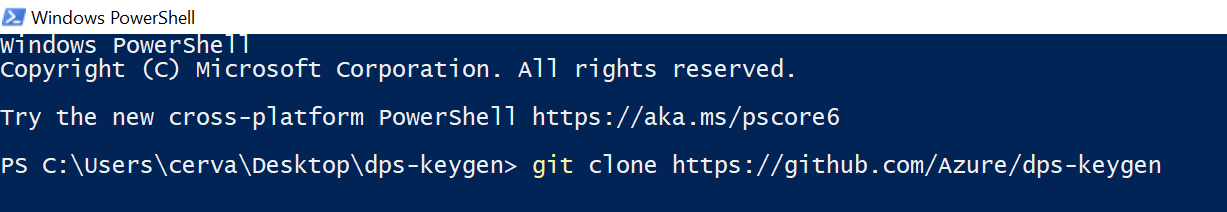
1. Create a folder named dps-keygen in your working directory



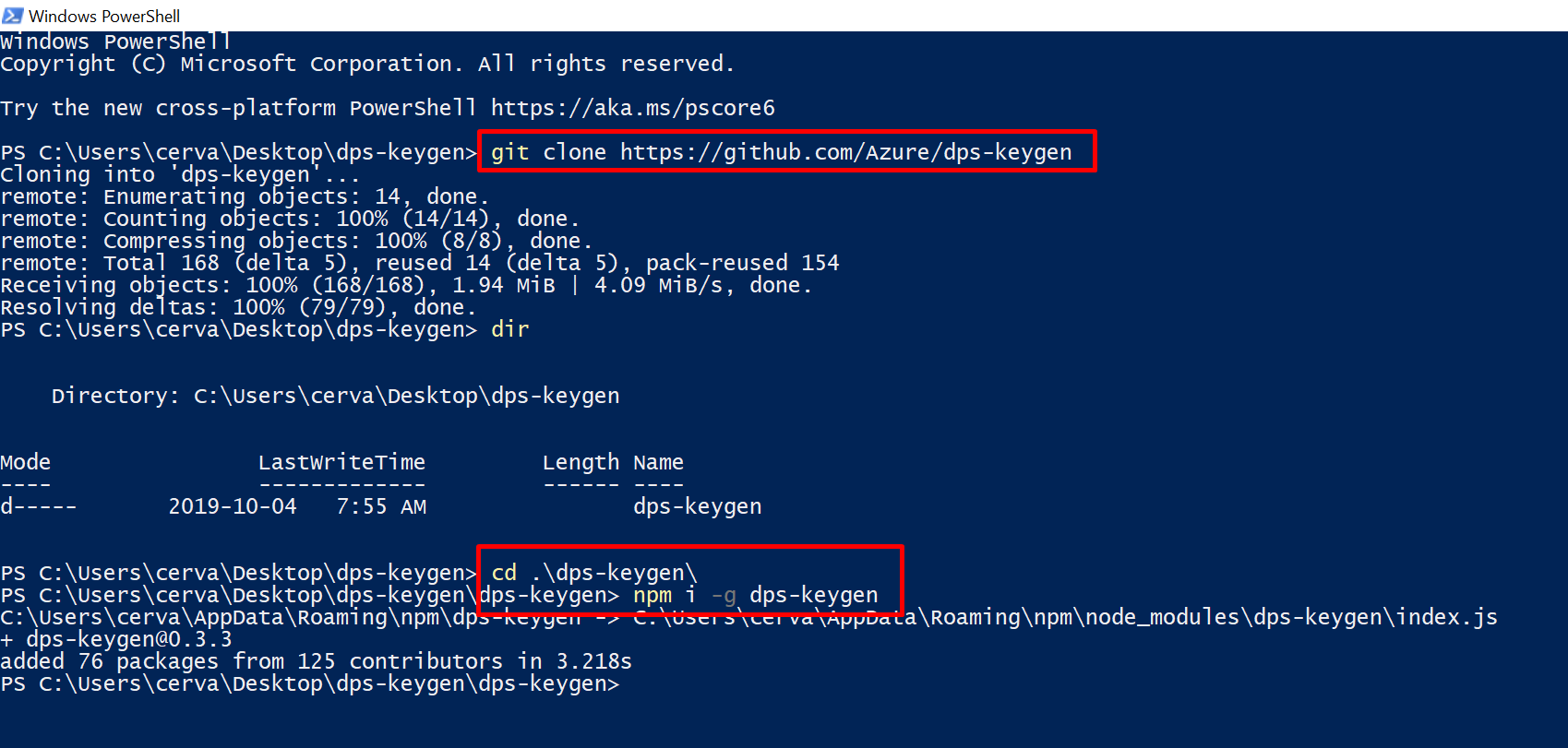
1. Run powershell in this directory



1. Clone the git repository: <https://github.com/Azure/dps-keygen>



1. Enter the dps-keygen folder you cloned, and run the npm command.



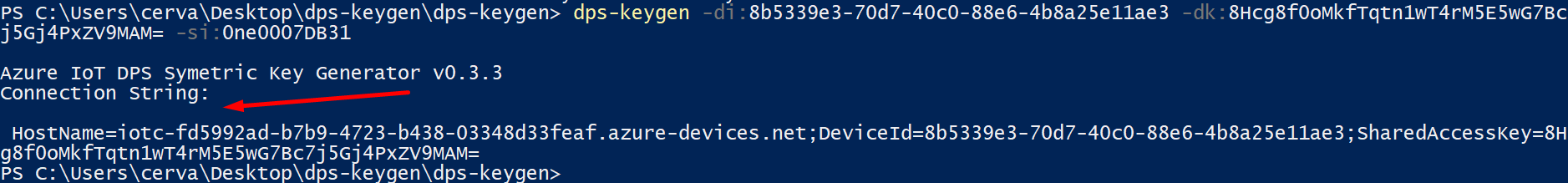
From the last step, we collected information like the below:

Scope ID: 0ne0007DB31  
Device ID: 8b5339e3-70d7-40c0-88e6-4b8a25e11ae3  
Primary Key: 8Hcg8f0oMkfTqtn1wT4rM5E5wG7Bc7j5Gj4PxZV9MAM=

Using the dps-keygen tool, we can run a function to get the connection string

dps-keygen -di:deviceidhere -dk:devicekeyhere -si:scopeidhere

dps-keygen -di:8b5339e3-70d7-40c0-88e6-4b8a25e11ae3 -dk:8Hcg8f0oMkfTqtn1wT4rM5E5wG7Bc7j5Gj4PxZV9MAM= -si:0ne0007DB31



**The provisioned connection string will look this this:** HostName=iotc-fd5992ad-b7b9-4723-b438-03348d33feaf.azure-devices.net;DeviceId=8b5339e3-70d7-40c0-88e6-4b8a25e11ae3;SharedAccessKey=8Hcg8f0oMkfTqtn1wT4rM5E5wG7Bc7j5Gj4PxZV9MAM=

# Step 5: Configure your IoT device for Azure

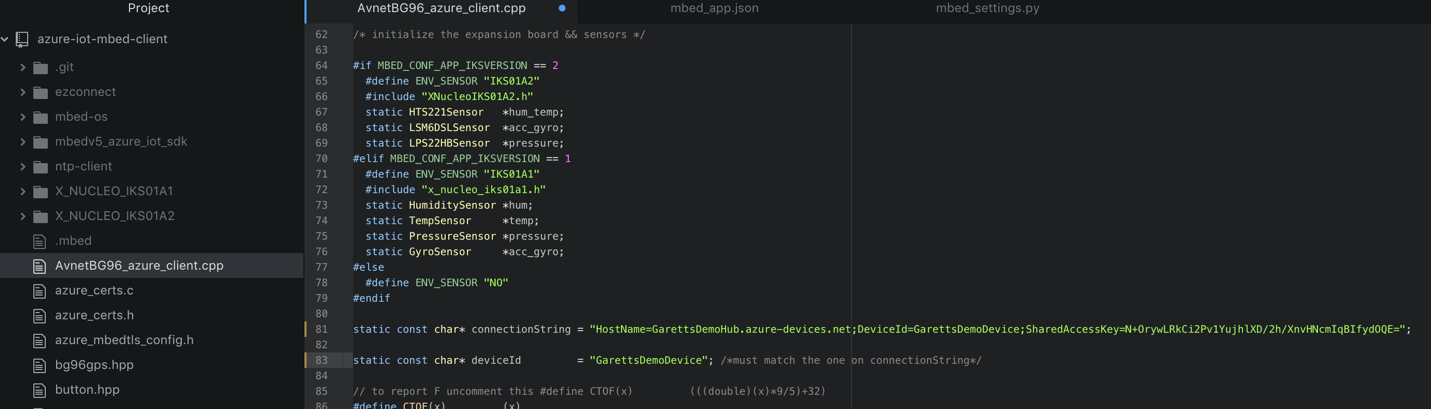
Getting back to the “Download the Avnet Azure IoT Client” step from earlier on in the tutorial, hopefully it has completed importing which should have created a folder for you named “azure-iot-mbed-client”, within this folder there are 2 different files we need to configure.

1. AvnetBG96\_azure\_client.cpp
2. mbed\_settings.py

**AvnetBG96\_azure\_client.cpp**

This file handles the sensor information gathering from the IoT board sensors, crafting the sensor data into a message payload and communicating that payload to Azure. In this tutorial we’ll leave the file logic pretty much as-is, but if you feel the need to modify the function of the board, I recommend looking back to this file at a later time.

The only thing we need to configure in this file is the name of the IoT device (deviceId, line 83) and setting the connection string (connectionString, line 81). Set the device ID to the name you used for the IoT device in Azure, and set the connection string to the “Connection String - primary key” we just copied a couple steps ago when creating the IoT device. One thing to note, the device ID is actually part of the connection string. Below is a screenshot of my configured file:



### 

### 

### 

### 

### 

### Formatting payload to accommodate IoT Central device templates

In the message sent to central, include another JSON object to have a location field with lat and lon. Edit IOTDEVICE\_MSG\_FORMAT on line 44 to match the below code:

#define IOTDEVICE\_MSG\_FORMAT       \

   "{"                             \

     "\"ObjectName\":\"%s\","      \

     "\"ObjectType\":\"%s\","      \

     "\"Version\":\"%s\","         \

     "\"ReportingDevice\":\"%s\"," \

     "\"location\":{" \

     "\"lat\":\"%6.3f\","     \

     "\"lon\":\"%6.3f\"},"    \

     "\"GPSTime\":\"%6.0f\","      \

     "\"GPSDate\":\"%s\","         \

     "\"Temperature\":\"%.02f\","  \

     "\"Humidity\":\"%d\","        \

     "\"Pressure\":\"%d\","        \

     "\"Tilt\":\"%d\","            \

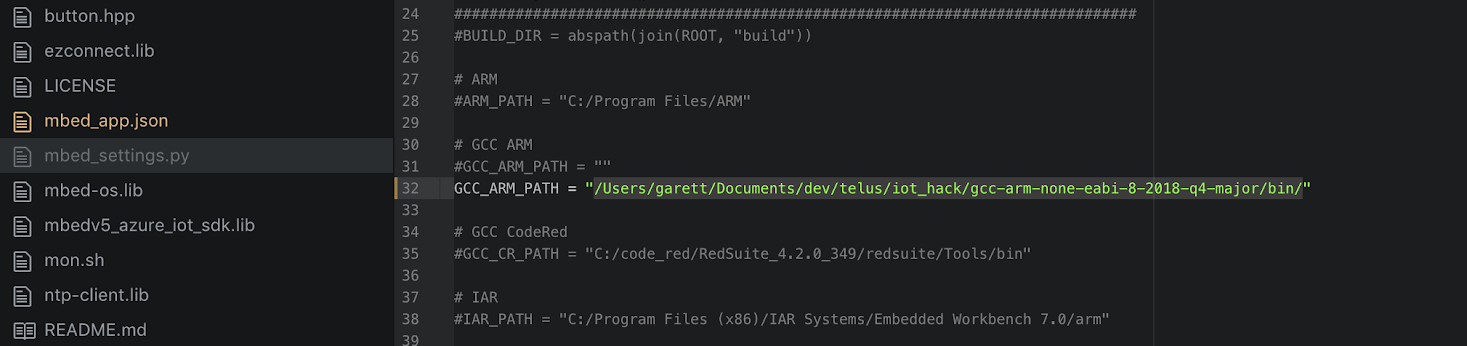
     "\"ButtonPress\":\"%d\","     \

     "\"TOD\":\"%s UTC\""          \

   "}"

**mbed\_settings.py**

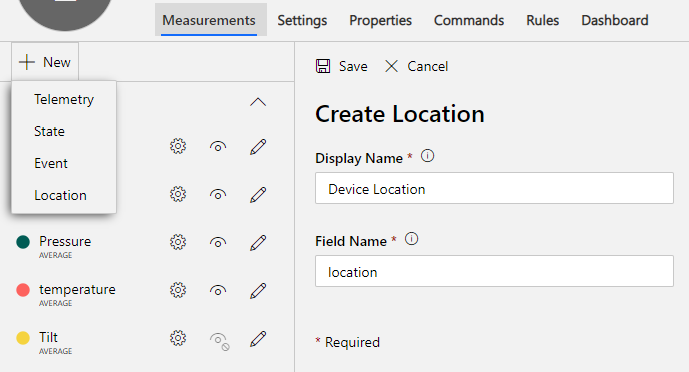
In this file we need to update the GCC\_ARM\_PATH value to the location where you extracted the “GNU ARM Embedded Toolchain”. In my case I changed the line from /usr/local/gcc-arm-none-eabi-7-2018-q2-update/bin/ to /Users/garett/Documents/dev/telus/iot\_hack/gcc-arm-none-eabi-8-2018-q4-major/bin/:



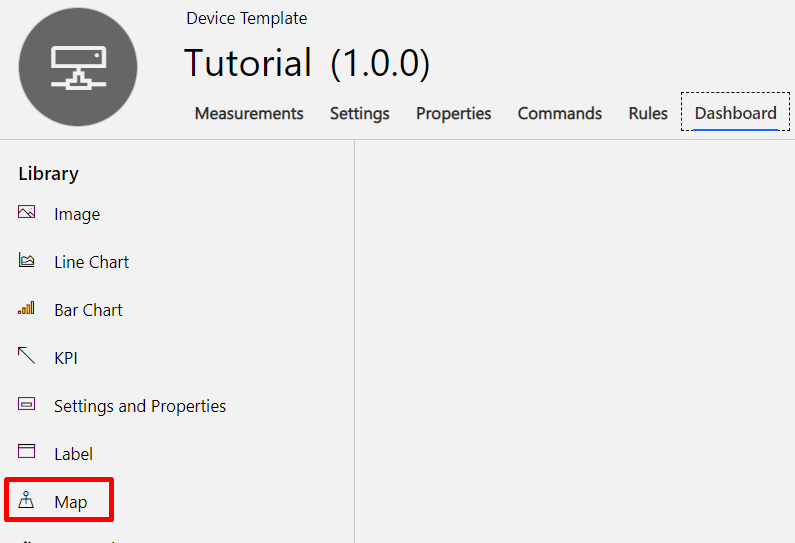
NOTE: Ensure you include the trailing slash, ‘/’ on a Mac, or compilation will not succeed!

Go back to IoT Central Application, on the go to “Device Templates” on the side panel, and select the template that your recently added device is using.

In Device template under Measurements, click New -> Location



In Device template under Dashboard, select Map.



In Configure Map, fill in the Title. For Location, under the drop down menu select “Location” under “Measurements”. Hit save.

## Compile Time!

If you’ve stuck with my rambling til now, I’m happy to say you’re now ready to compile the Azure client and get it loaded to your IoT device. The following steps will get your client compiled and loaded to your board:

1. Run the terminal or command-line on your Mac or Windows PC respectively
2. Change to the directory to azure-iot-mbed-client (this is created in the same directory where we ran mbed import above)
3. Install the required Python packages by running the command:

* python -m pip install -r mbed-os/requirements.txt
* If you encounter errors, try appending --user to the abve command and re-run

4. Plug a USB cable from the L496 MCU (white board) using the micro-usb cable into your computer

5. Check to see if there is a USB drive detected called NODE\_L496ZG. This means your board is connected.

6. Run the command:

* mbed compile -m NUCLEO\_L496ZG -t GCC\_ARM --profile toolchain\_debug.json
  + *You may need to prepend the command with python -m on Windows or use sudo on Mac*
* If you get “error: Could not find executable for GCC\_ARM.” make sure the path you set in mbed\_settings.py starts from your azure-iot-mbed-client directory
* If you get an error about missing dependencies, you can try:
  + check if Git is at the latest version. If not, update it.
  + Re-pull all pre-requisites that was pulled using git.

1. If all goes well, you will see the mbed compiler start creating your new bin file. When it is complete, the file can be found here, relative to the azure-iot-mbed-client directory you should still be in: BUILD/NUCLEO\_L496ZG/GCC\_ARM/azure-iot-mbed-client.bin
2. Drag the created binary over to the NODE\_L496ZG drive, this will load the new client software and reboot your IoT board

Once your board reboots it will immediately attempt to connect to the network, read sensor data and send that data to your IoT Hub.

**Monitoring Data**

If all goes well, your hub will start receiving the data from your board without incident. If any issues arise, or you just want to have a better idea of what is being sent to your Hub, it would be helpful to be able to see what exactly your board is doing and the raw data being sent.

**See Your Board Status**

With the IoT board connected to your computer you are able to analyze the board status using the COM port the board has connected to the computer using.

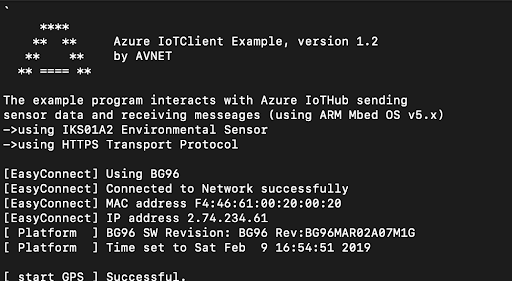
MacOS

1. From your terminal and issue the command ls /dev/tty.\* This will show all the serial ports you have. Look for /dev/tty.usbmodemxxxxx (on my Mac it was 14203), which will be the board
2. Issue the command screen /dev/tty.usbmodemxxxxx 115200 (where xxxxx is for your particular Mac). This connects to your device and displays the terminal output with baud rate of 115200.

Windows

1. Download and install the [Quectel LTE USB Driver](https://github.com/TELUS-Emerging-IoT/TELUS_IoT_Getting_Started/blob/master/files/Quectel_LTE_Windows_USB_Driver_V1.0.zip)
2. Using your client of choice (I prefer [Putty](https://www.putty.org/)) open a Serial connection to the COM port the board is using (you can determine this using Windows’ Device Manager), and a Baud Rate of 115200.

If you don’t see anything in the terminal after following the above steps, press the black “RESET B2” button on the white board, this will reboot the board and should present you with a screen similar to this one in the terminal:



Output will continue to produce as the board makes repeated network sends to Azure. You won’t, however, get to see the actual payload being send.

**Monitoring Payloads Sent to Azure**

The Azure CLI tool will let us monitor the payloads sent from the board to Azure. The following commands will let you see the payloads sent in real-time:

1. Issue the following command to log in to Azure from the command-line

* az login
* A browser will open, log in using your Azure credentials

1. Install the Azure IoT extension:

* az extension add --name azure-cli-iot-ext

1. Retrieve the “Connection String - primary key” that you copied earlier when you created your IoT Hub, with it, issue the following command in the command-line terminal:

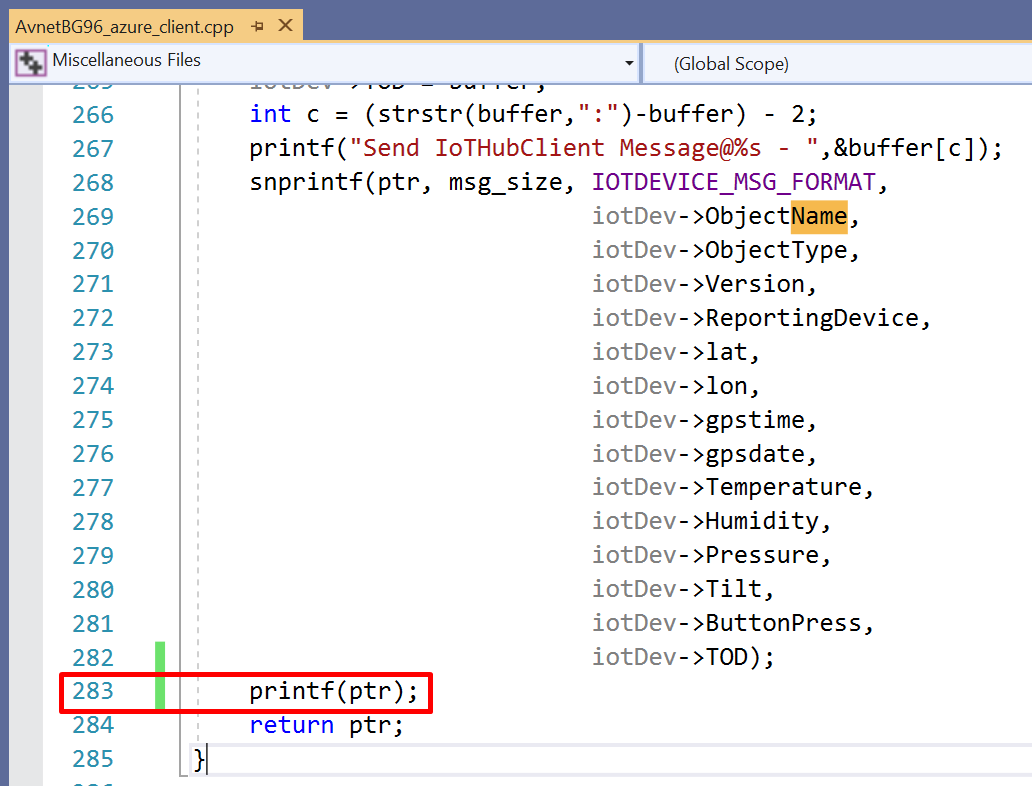
* az iot hub monitor-events --login "<your\_connection\_string"

If all goes well you will start seeing JSON payloads as they are sent to the server

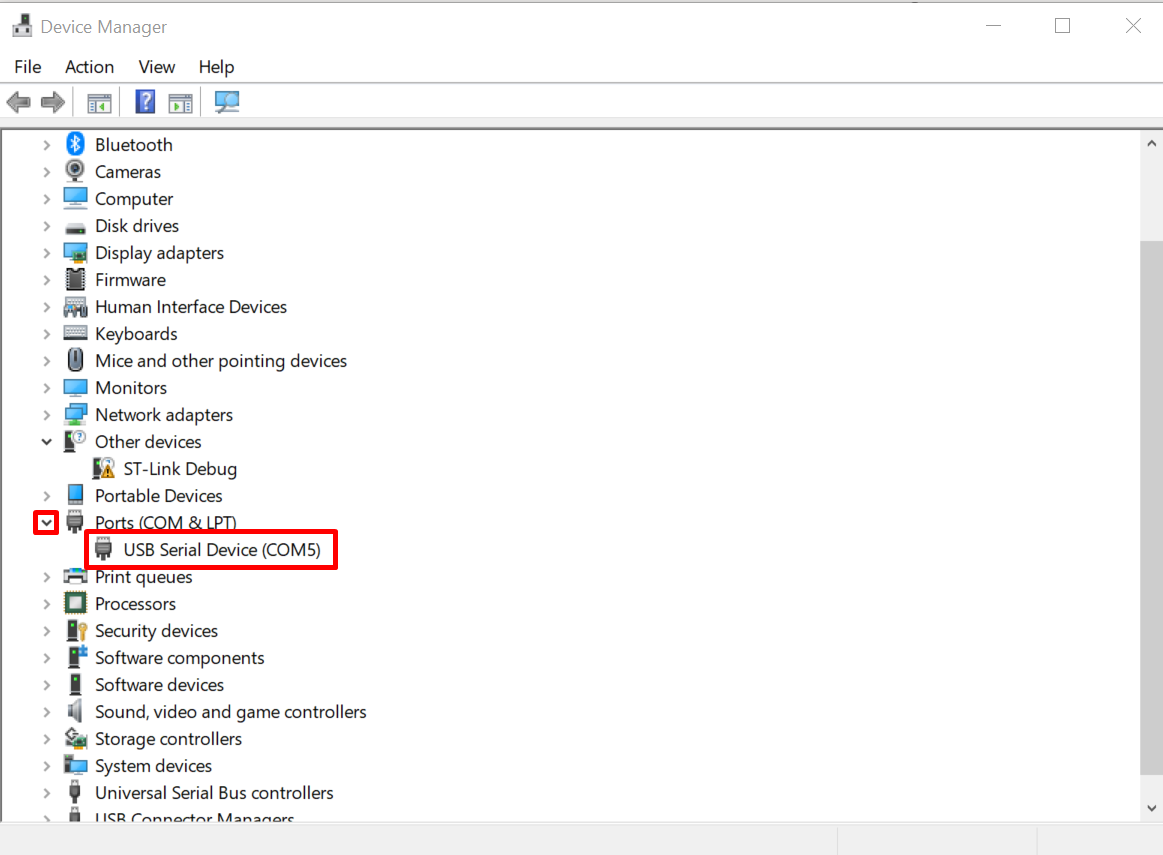
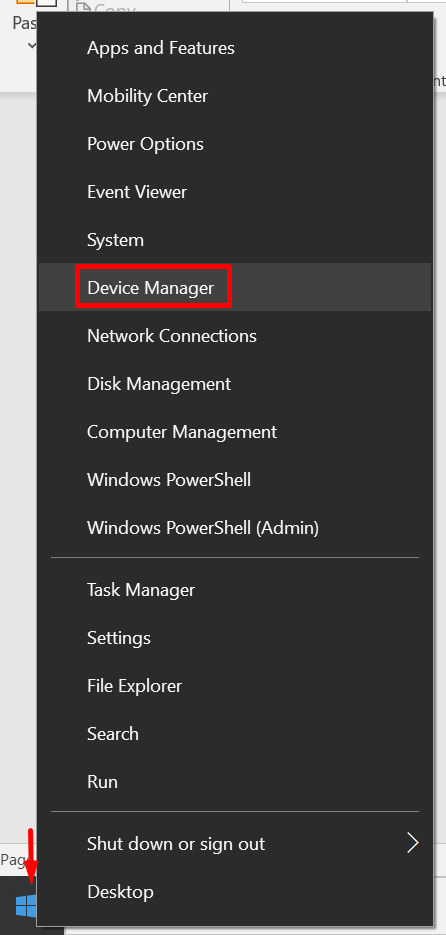
# Step 6: Viewing the data being transmitted and received

### Data being sent mirrored in the port (while PC is connected)

You can add instructions in the AvnetBG96\_azure\_client.cpp to print out information to stdout after you finish loading the payload for sending. To do this, simply add a printf immediate after an item is made.

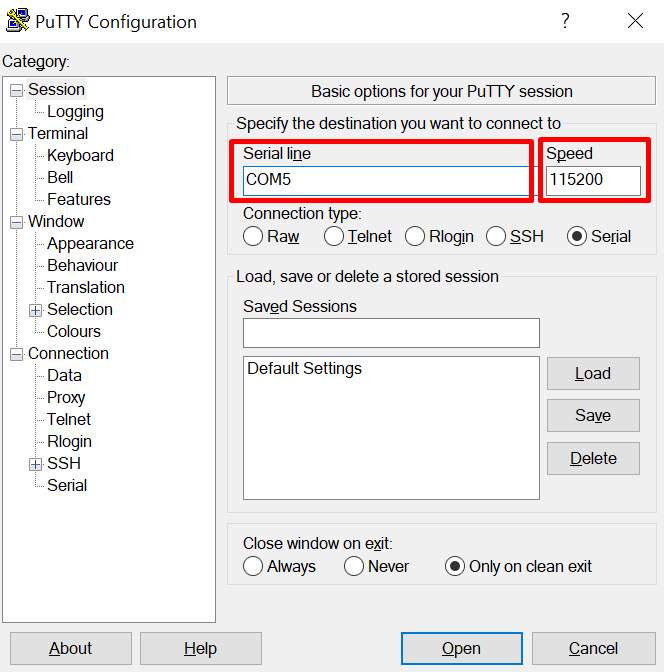


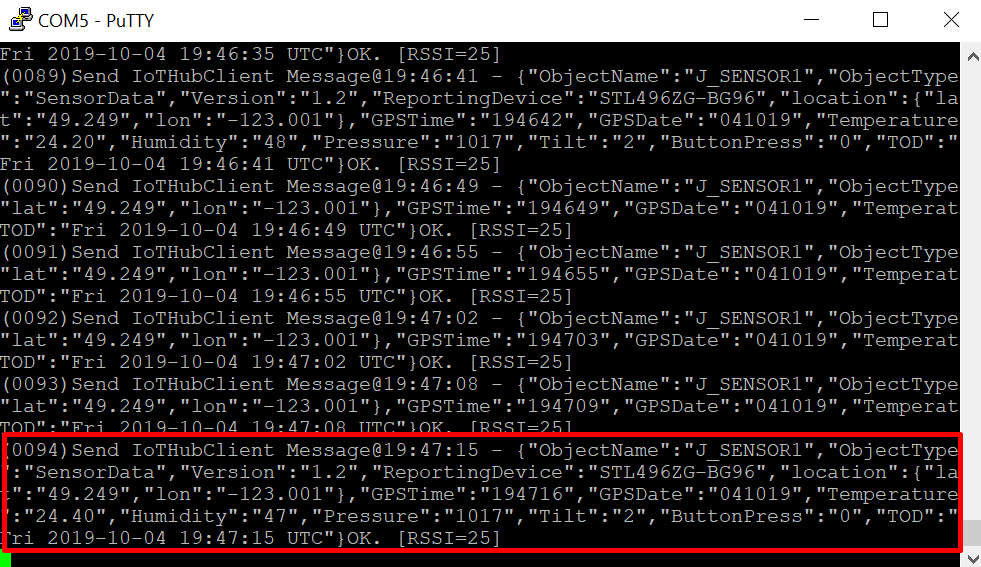
Once you add the printf command, start monitoring your port. [Putty](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) is a program which enables you to do this. Take note of the port your device is currently attached to.





Open Putty, change to the port which was observed in your device manager, and set the speed to 115,200.



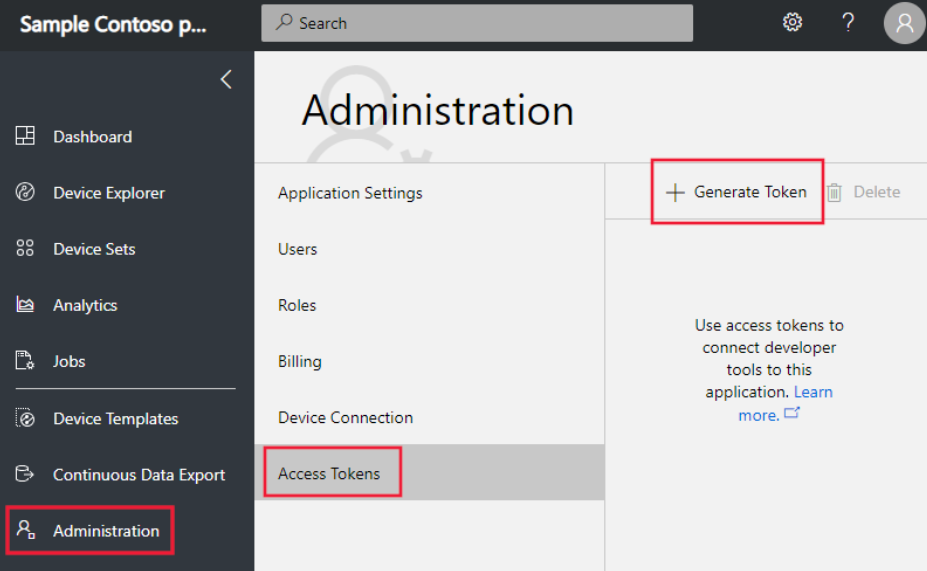


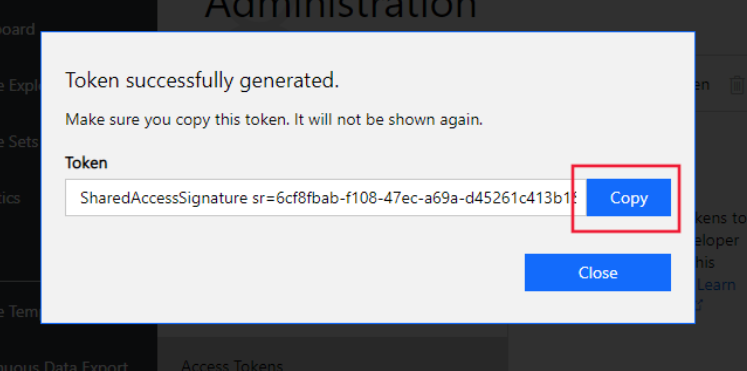
### Data received by Azure

<https://docs.microsoft.com/en-us/azure/iot-central/howto-use-iotc-explorer>

Run the following command from any CLI (node must be installed): npm install -g iotc-explorer

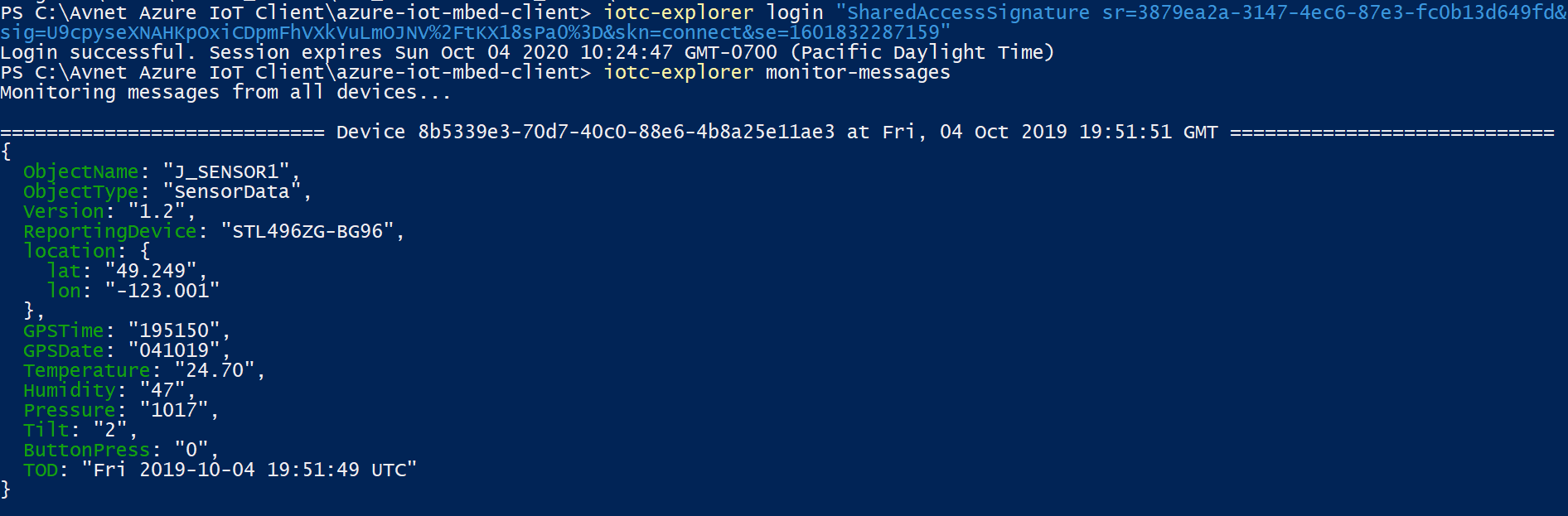
Navigate to Administration then Access Tokens. Generate a token.



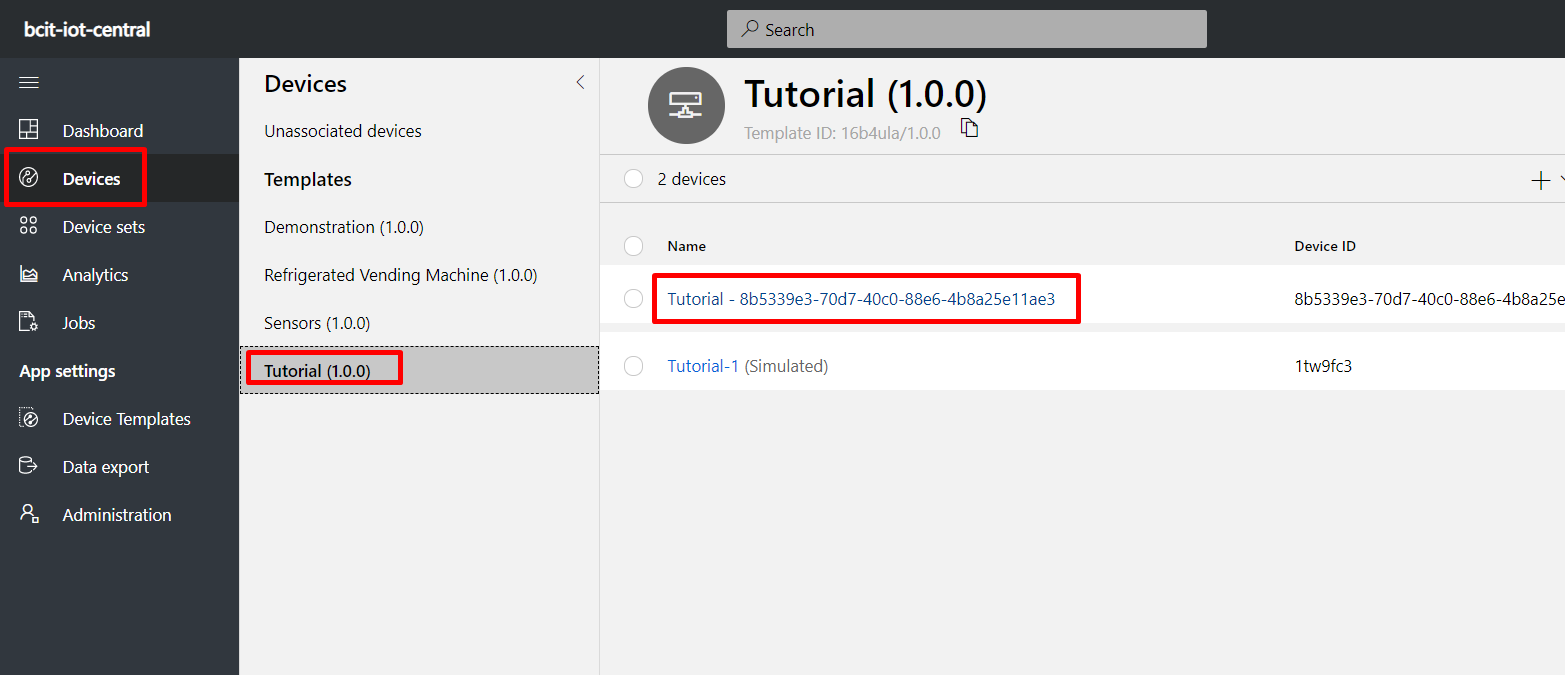
Copy access token dialog screenshot

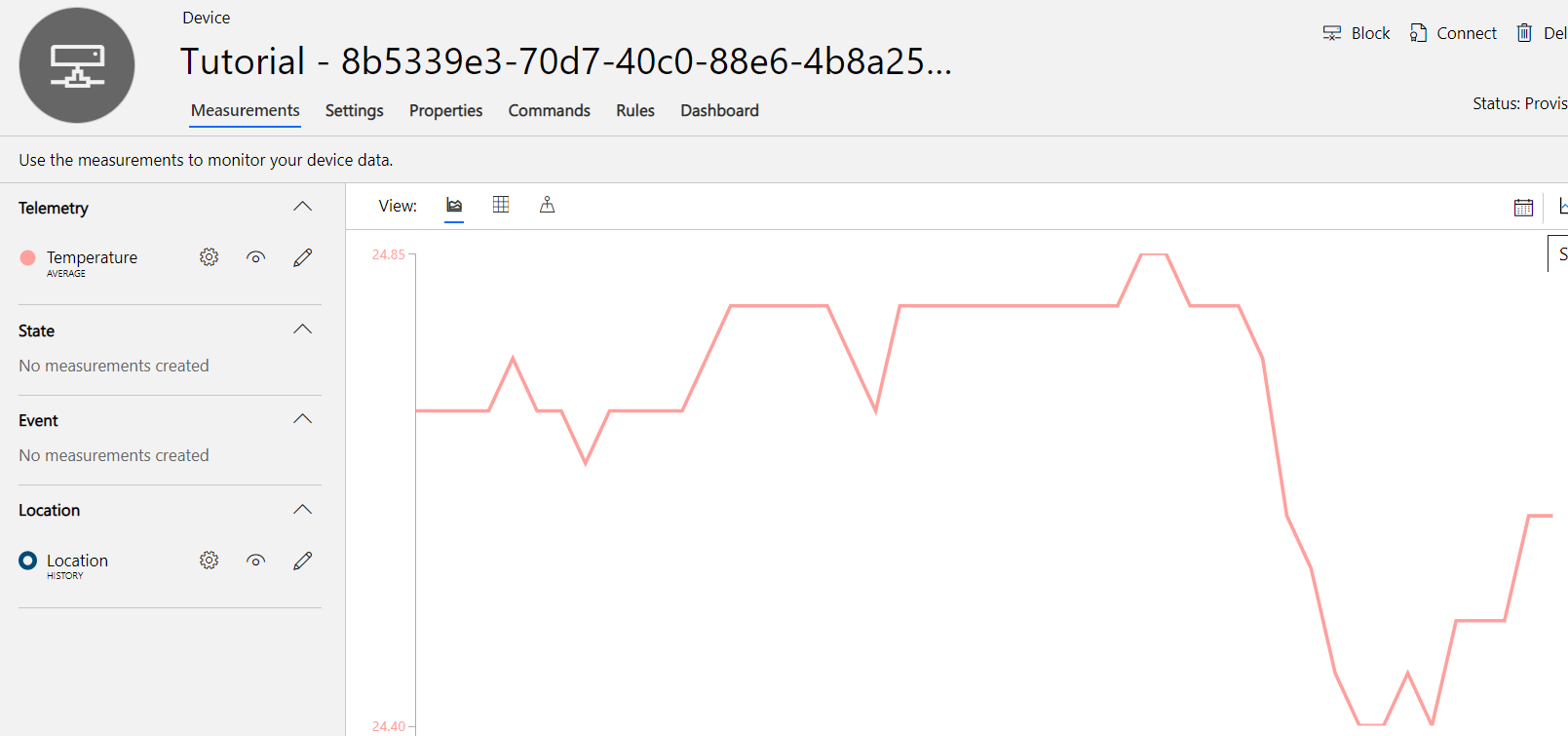
You can use the token to log in to the CLI as follows iotc-explorer login "<Token value>"

Running the command iotc-explorer monitor-messages will then monitor what Microsoft Azure IoT Central is receiving.



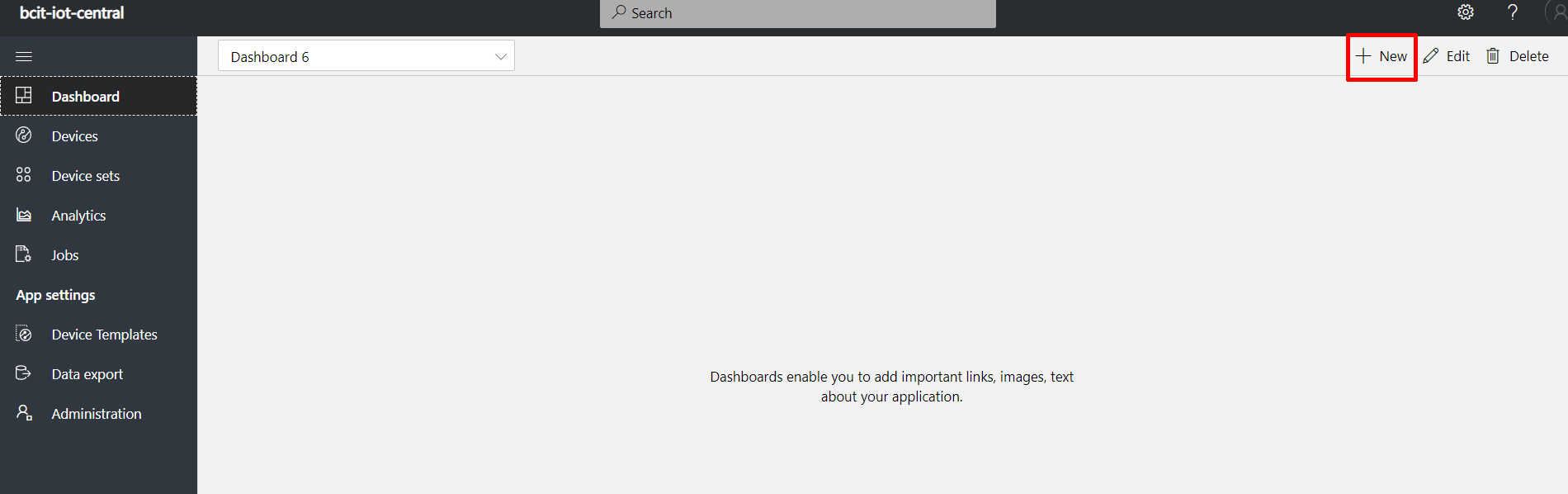
### Data received in Azure visualized



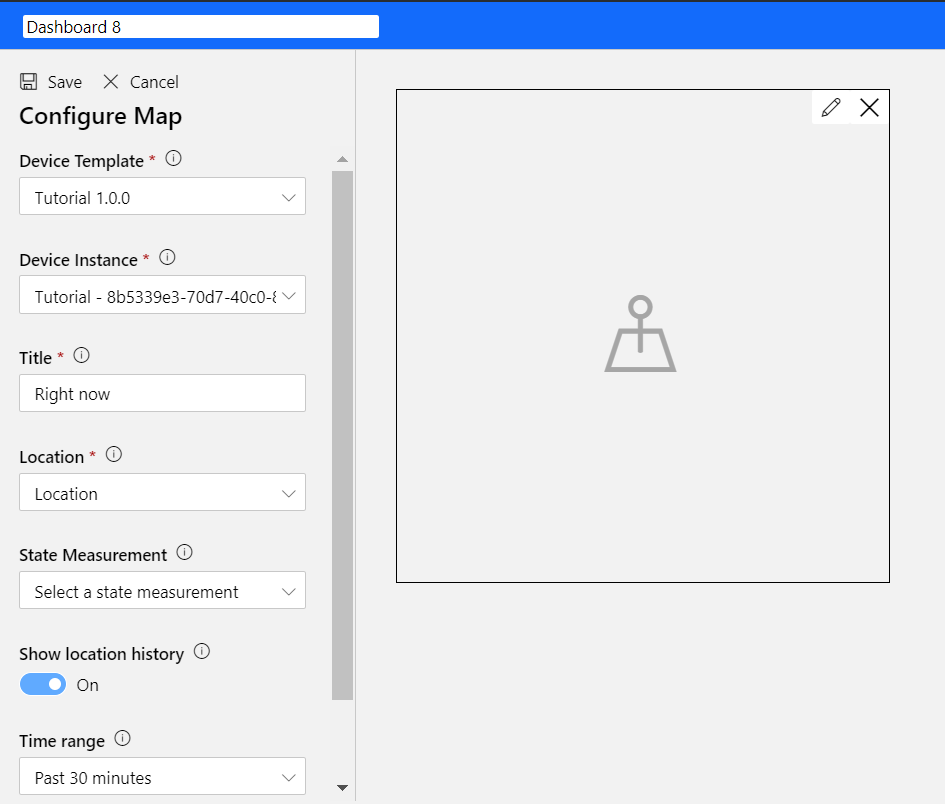


# Step 7: Adding widgets to your dashboard

Select a new widget from the dashboard menu



As a test, we’ll add a map.



That’s all there is to it! Keep adding widgets and continue building out your board.

