

Internet of Things Workshop

Lab 2

Connect Device to IoTHub

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
| 9/18/2016 | Steve Busby | 1.0 | Initial draft |
|  |  |  |  |

Table of Contents

[Change Record 2](#_Toc461796314)

[Introduction 4](#_Toc461796315)

[1.) Create a Stream Analytics Job 5](#_Toc461796316)

[2.) Defining the Input 7](#_Toc461796317)

[3.) Creating the output EventHub queue 9](#_Toc461796318)

[4) Defining the Stream Analytics Output 13](#_Toc461796319)

[5) Writing the Streaming Query 15](#_Toc461796320)

Introduction

The purpose of this lab is to get an IoT device connected to and talking to Azure IoTHub and the RM-PCS. While this is demo device in a demo situation, it is very representative of the process of connecting any device to Azure. We will leverage the Raspberry PI 3 and a DHT22 temperature and humidity sensor to represent our device. The connection to Azure will leverage a Python script running on Linux, which is a very common IoT development environment. However, it is certainly not the only option. The Azure IoT Device SDKs leveraged for this lab are also available in the following languages:

* C
* Java
* C#
* Python
* Node.js

It has also been tested and used on these platforms

* Linux
* Windows (universal – desktop and mobile)
* Various Real-Time OS’s (RTOS), including FreeRTOS, Arduino, and many others
* IoS
* Android.

In this series of labs, you will:

1. Create and navigate the Azure IoT Remote Monitoring Pre-Configured Solution (RM-PCS)
2. Create a device to read a temperature and humidity sensor (and optionally, a light sensor) and send that data to the RM-PCS for display
3. Create a Stream Analytics job that looks for ‘high temperature’ alerts and outputs that alert to a queue for further processing
4. Create an Azure Function that takes that alert, and sends a command to the device to turn on or off an LED depending on the alert condition.

At the end of this lab you will have a physical IoT device connected to Wifi, sending telemetry data to Azure IoT, and listening to and responding to commands from Azure.

1. Environment and device setup

In this section, we will get our desktops talking to our Raspberry Pi’s (RPI), login, and make sure they are connected to the network. Once done, we will make the physical connections between our Pi’s and the DHT22 temperature and humidity sensor, and an LED to represent a “high temperature alarm”.

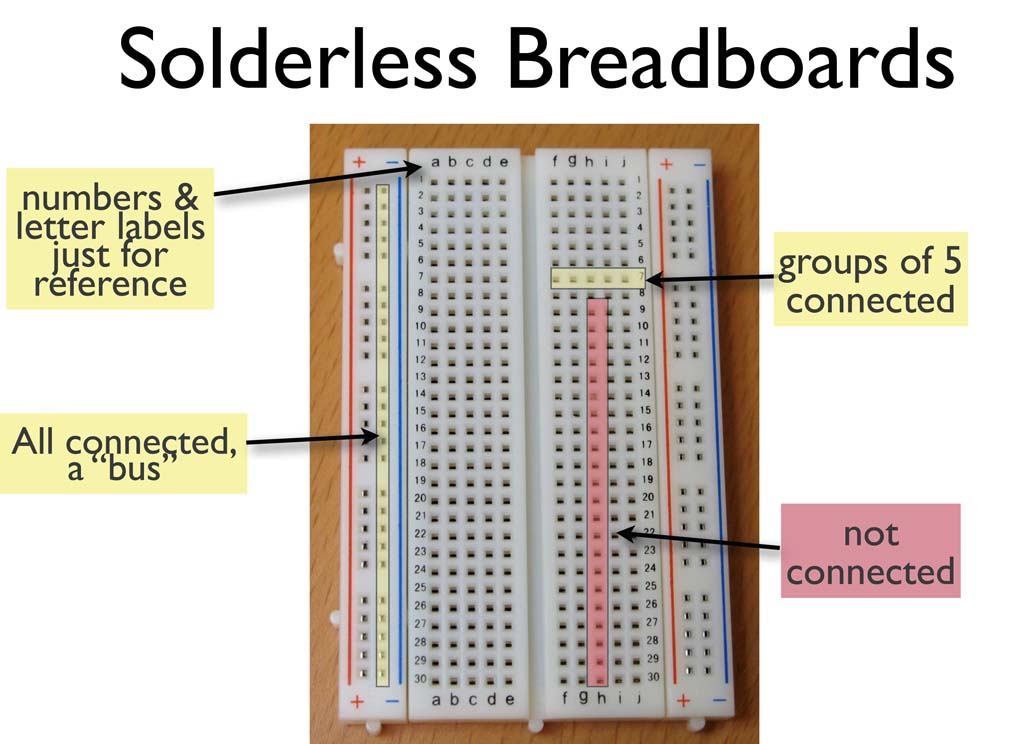
Several things have already been pre-setup for you on the RPI’s provided for this lab

* Raspbian Jesse, a version of Debian Linux has been pre-installed
* Serial communication has already been enabled (so you can ‘console’ into them over a serial connection)
* They have been pre-connected to J&J’s guest wireless
* The Azure IoT SDK has already been downloaded from github ( <https://github.com/Azure/azure-iot-sdks> if you want to check them out yourself)
* Python drivers for the DHT22 sensor have been preinstalled (<https://github.com/adafruit/Adafruit_Python_DHT>)

If you get this lab manual outside of the workshop delivery for which it was developed (meaning none of this stuff has been done for you ☺), the process for setting up the RPI for the lab is in Appendix A

For the lab, we’ll first wire up the RPI before we power it on and connect to it. Like with most electronics, it’s always safer to do any wiring with the device powered off. If your RPI is powered on, remove the power at this point.

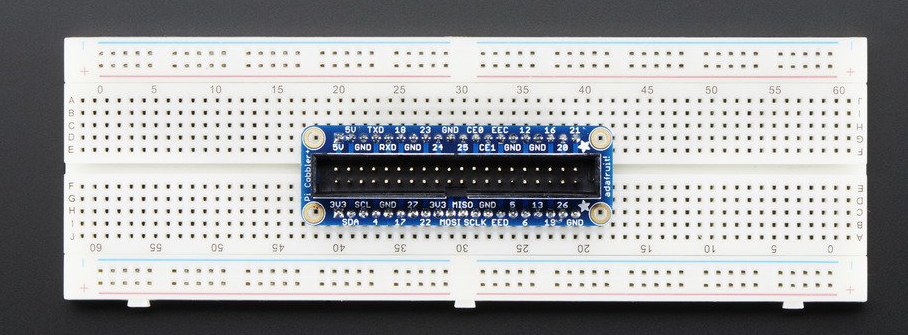
For our connections, we will be using a solderless breadboard for connecting the DHT22 temperature sensor to the RPI. If you aren’t familiar with a breadboard, they work like this:



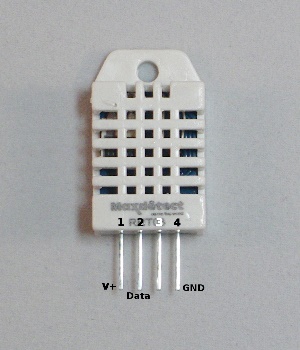
On a breadboard, the side rails (labeled + and -) are electrically “connected” all the way down the sides. They are generally used for power (+) and ground (-) or GND. In the middle section of the breadboard, each row of 5 connections are electrically connected together (but adjacent rows are not). So, for example, following the grid on the picture 1A, 1B, 1C, 1D, and 1E are connected together, but none of those pins are connected to either 2A, or 1F.

First let’s cover the electrical components involved.

Your kit comes with what’s called a “cobbler”. The purpose of the cobbler is to bridge the connections between your RPI and the breadboard. The cobbler plugs into the breadboard (as shown below), and then connects via the supplied ribbon cable to the ribbon connectors on the RPI



The DHT22 temperature and humidity sensor has pins that will plug into the breadboard directly.

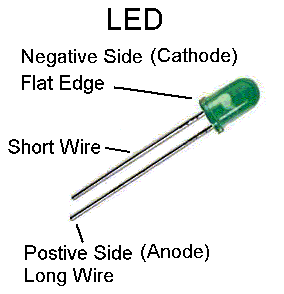


With the sensor facing you, like shown, the pins from left to right are:

1. V+/VCC – power. We will connect this to 5V power
2. DATA – this is the pin we will read the data from with the RPI
3. UNUSED – this pin is not used
4. GND – Ground. This pin will be connected to ground

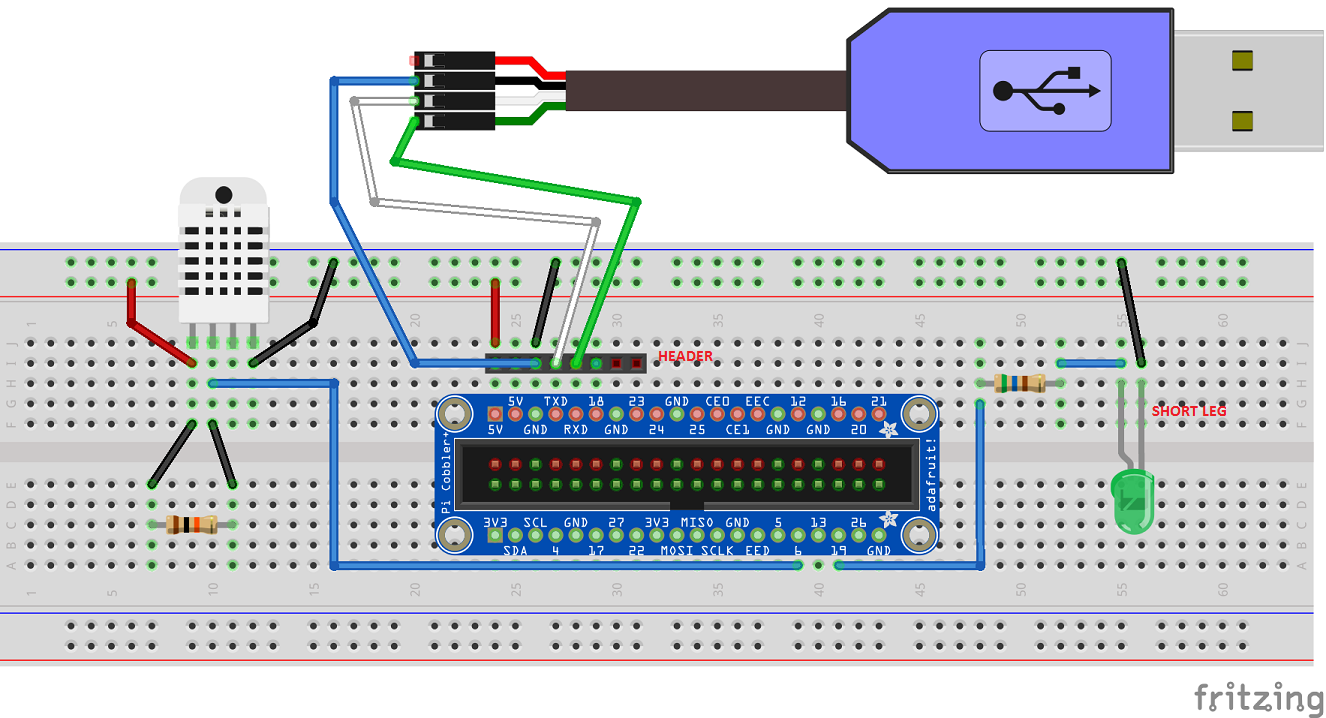
To keep the data pin from “floating” when it’s not being driven by either the RPI or the sensor, we will connect a 10k Ohm “pull up” resistor between the DATA pin and the power pin. This will ensure that, if the pin is in between data bits being transferred, it gets “pulled up” to 5V. This helps cut down on false or bad readings from the sensor.

And finally, we will use a light-emitting diode (LED) to represent our “high temperature alarm” indicator. In addition to lighting up, LEDs only allow power to flow in one direction, so we need to make sure they aren’t connected backwards. At these voltage levels, you won’t hurt the LED if you hook it backwards, it just won’t light.



As you can see from the picture, an LED has a positive side (anode) and a negative side (cathode). You can tell the difference because the cathode is the shorter of the two legs. The cathode must be connected to the “ground side” of the circuit with the anode connected to the “power side”. There may be other components of the circuit before or after the LED (in our case, a resistor), but the cathode in our case, will be connected to ground/GND. Finally, to keep the LED from drawing too much power from the RPI (the pins on an RPI can only ‘source’ a limited amount of power), we will connect a 560 Ohm resistor between the anode of the LED and the RPI pin that we use to drive it.

The picture below shows the connections that need to be made between your RPI cobbler and the different components, including the USB/TTL console cable (the USB cable with female wiring ends)



A few notes about the connections

* The resistor that is connected to the DHT can be placed directly between pin 1 and pin 2 of the DHT sensor instead of having it in a separate part of the breadboard and wires running. My drawing tool just wouldn’t let me show it that way
* It’s hard to see, but the USB/TTL cable, because it has female ends, is connected to a header that is stuck in the breadboard (which sticks up with male ends). This is a picture:



* Also, hard to show in the diagram, but you don’t need to connect wires from the USB/TTL cable to the headers, you should just plug the cable ends directly into the header

That’s it for the connections. If wired wrong, you can damage the DHT22 sensor, so at this point, before you fire up your RPI, feel free to call on the proctor(s) to double check your wiring. You don’t want to ‘let the magic smoke out of the box’ ☺

2.) Connected to and powering your RPI

At this point, we are almost ready to power up our RPI. Before we do, let’s connect our console cable so we can watch the PI boot.

While our RPI will connect over Wifi to the Internet (and Azure), and our laptops will also connect over Wifi to the Internet on the same network, corporate policy will not allow you to SSH from your laptop to the RPI directly. So, we must connect over serial console with the USB/TTL cable.

To use the cable, we must install a driver, and do some configuration (and install a terminal client, in the case of Windows).

To set up the connection, follow the instructions (with the caveats below) at <https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable?view=all>

Caveats:

* The serial console has already been enabled on your RPIs, so you can skip down to the software installation section about mid-way down the page
* Make sure you get the right drivers for your machine (PC vs. Mac)
* For the actual connection of the console cable to your RPI, follow the picture in the previous section, not the instructions on the site
* Do NOT connect the red cable from the USB/TTL cable. We will be powering our RPI externally.

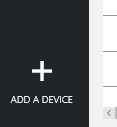
Once the PI is up and going. Log in with username ‘pi’ and password ‘raspberry’. To check network connectivity, try ‘sudo ping 8.8.8.8’ (CTRL-C to stop).

Congrats! You and up and running and ready to talk to Azure!

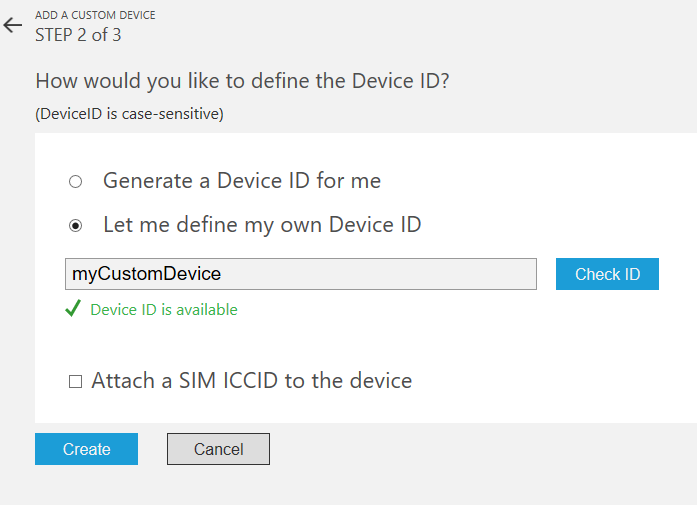
3.) Gathering IDs, Keys, and Connection Strings

Before we can connect the device to the Azure RM-PCS, we need to let the solution know about the device (so we can authenticate).

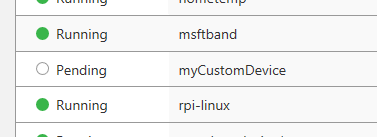
1. Navigate to your RM-PCS solution (https://<solutionname>.azurewebsites.net)
2. On the bottom left corner, click the “Add a device” button



1. Click Add New under “Custom Device”
2. On the next screen, change the radio buttons to “let me define my own Device ID”, and pick a deviceID for your device



1. Click Create
2. Your device is now added to the RM-PCS. Copy the three parameters displayed on this page, Device ID, IoTHub Hostname, and Device Key. Paste them into notepad, as we will need them in the next step
3. Click Done. On the Devices screen, note that your device is in a “pending” state. That essentially means that the solution knows about your device and is ready for it, but has not yet “heard from” that device. We will remedy that in the next section



4.) Posting Telemetry data to Azure

The code to read from your DHT22 sensor and post to Azure has been provided as part of this lab. Now that we have our keys, strings, etc, we’ll need to download it, modify it for your specific RM-PCS solution details, run it, and test it. Follow the steps below to get started

1. On the RPI, we need to download the code. We’ve provided the python script, and also pre-compiled the python SDK for your use. If you are interested in the steps to compile the python module yourself if you need to start from scratch, see the details in Appendix B
2. To download the code, enter these commands on the RPI
   1. cd ~
   2. git clone –recursive <https://github.com/stevebus/IoTWorkshop>
   3. cd IoTWorkshop/Lab2
3. the code is downloaded, now we need to put the details for your specific device, RM-PCS, and (just for fun), location
4. edit the lab2.py script with your favorite linux editor. If you don’t have one, use nano
   1. enter “nano lab2.py”
5. Read through the code, which is well commented, and make sure you understand what it is doing.
6. scroll down to line 48 (you can see what line you are own by typing “CTRL-C”). Line 48 is where you enter the connection details you got from the “add device” process in the previous step. Copy/paste your host name, device id, and device key you copied into the placeholders (make sure to delete the < and > signs). Full screening ‘putty’ allows you to see more of the line at once.

connection\_string = "HostName=<host\_name>;DeviceId=<device\_id>;SharedAccessKey=<device\_key>"

1. scroll down to line 155. This is the line that contains the “DeviceInfo” string that describes your device to the RM-PCS solution

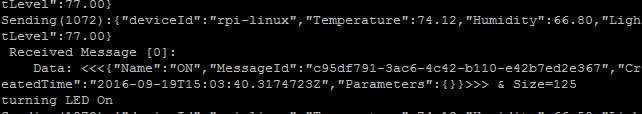
deviceInfoTxt = "{\"ObjectType\":\"DeviceInfo\", \"Version\":\"1.0\",\"IsSimulatedDevice\":false, \"DeviceProperties\":{\"DeviceID\":\"rpi-linux\", \"HubEnabledState\":true,\"Latitude\":33.620325, \"Longitude\":-86.968372},\"Commands\":[{ \"Name\":\"ON\", \"Parameters\":null},{ \"Name\":\"OFF\", \"Parameters\":null}]}"

1. go to <http://www.bing.com/maps> and search for your home, your home town, or some other location that interests you. On the map you can see the latitude and longitude. Substitute those values into the line above (you’ll need to arrow out to the right to get to it).
2. Save the file. If you are using nano as the editor, hit CTRL-O, <enter>
3. Close the file (CTRL-X in nano)
4. Execute the script (‘python lab2.py’ from the command prompt)
5. You should see the DeviceInfo string echo’ed to the screen and sent to the IoTHub, and then every 3 seconds, you should see the temperature and humidity (in a JSON string) sent to the IoTHub. The LED should also briefly flash to indicate we are sending data
6. Congratulations, you’ve connected a physical IoT device to the Azure IoT RM-PCS. Next we can look at the telemetry, as well as test manually sending a command to the device from the portal, which we will do in the next step

5.) Posting Telemetry data to Azure

Now we can take a look at the RM-PCS portal and make sure everything is working before we move on to the next lab.

1. Navigate to the RM-PCS portal for your solution (https://<solutionname>.azurewebsites.net)
2. Look on the map and you should see your device. You can either click on that device, or from the “Devices to View” drop down, you can select your device. You should see the temperature and humidity displayed there, and it should match the data you see flowing from the RPI via your putty console
3. Feel free to hold your fingers over the DHT22 sensor, or breath on it, to vary the temperature and humidity and watch the values change on the portal
4. After looking at the telemetry data, click on the “Devices” tab on the left hand nav
5. Choose your device from the list by clicking on it. Once the Device Details pops out from the right hand side, choose “Commands” to go to the commands page.
6. From the Select A Command box, you should see the two commands that we told the RM-PCS we support via the DeviceInfo message we sent from the device, “ON”, and “OFF”.
7. Choose “ON” from the drop down and choose “Send Command”. The LED on the device should light, and you should see this message appear in your RPI console



1. Now select the “OFF” command from the drop down and send the command. Observe the LED turn back off and the corresponding message in the RPI console

Congratulations! You now have a physical IoT device talking to the RM-PCS. In the next couple of labs, we’ll do some processing of the telemetry data looking for high temperature “alarms” and responding to them.