

Internet of Things Workshop

Lab 5

Integrating with Azure IoT

Suite Remote Monitoring

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
| 3/2/2016 | stevebus | 1.0 | Initial draft – specific to the Otis Elevator delivery (students are not admins in the RM solution) |

Table of Contents

[Change Record 2](#_Toc444689539)

[Introduction 4](#_Toc444689540)

[Environment setup 4](#_Toc444689541)

[1. Configuring the IOT Hub 5](#_Toc444689542)

[2. Setting up the gateway device 9](#_Toc444689543)

[3. Retrieving the serial port ‘name’ 10](#_Toc444689544)

[4. Coding the gateway 12](#_Toc444689545)

Introduction

This lab is focused on integrating the IoT device/gateway created in previous labs with the Azure IoT Suite Remote Monitoring Pre-configured solution. For the lab, you will be integrating with an existing remote monitoring pre-configured solution, however, outside of the lab environment you can generate your own pre-configured RM solution at <http://www.azureiotsuite.com>

In this series of labs you will:

1. Assemble an Arduino Uno device for temperature monitoring using a prototype kit, and code and deploy a sketch using the Arduino IDE.
2. Write a gateway application (Universal Windows App) on a Raspberry PI to receive the serial data from the Arduino and send data to an Azure IoT Hub.
3. Configure Azure Stream Analytics jobs for gathering and aggregating streaming data for reporting purposes.
4. Build a Power BI dashboard for visualizing real-time and historical event data from the sensor.
5. Integrate the gateway app with the Azure IoT Suite Remote Monitoring pre-configured solution.

At the end of this lab you will have changed the gateway code developed in Lab 2 to point to an instance of the Azure IoT Suite Remote Monitoring solution.

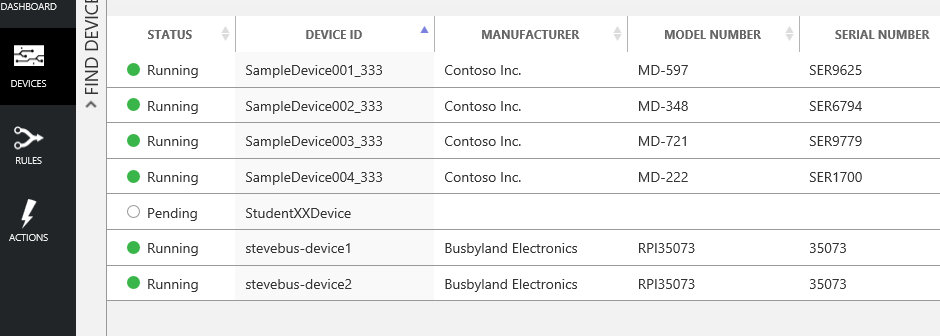
Environment setup

You must have an instance of the RM solution deployed. For this lab delivery, we assume you already have one (provided for you).

1. Review the device information in the pre-configured solution

In this section, you will look at the pre-configured solution and view your device, including the keys, etc.

1. Navigate to the URL provided by your class instructor to the RM instance
2. Note the dashboard shows a few existing devices and has a graph for their telemetry data (temp and humidity)
3. Click on the devices tab on the left.
   1. You will see a number of devices, some of which have metadata and a “running” status and several (one per student) that have a “Pending” status. That means that a device ‘record’ has been created in the solution for the device, but the device has not yet connected and sent it’s meta data (or telemetry). We will remedy that shortly!.



* 1. Click on your device (per your student ID) and note the device details on the right. Note that the details are sparse at this point, as the device has not yet connected and self-reported its details.

1. Update the gateway code

In this section you will edit the device connection settings to point to the IoTHub that supports the RM solution, as opposed to your own. You will also add a new class and function to encapsulate the “DeviceInfo” for your device and ‘update’ the RM solution with your device’s metadata. This allows the device to describe itself, including the commands it supports, to the RM solution.

1. Open the StudentXXGateway project from Lab 2 and open the StartupTask.cs file
2. From the lab instructor, obtain the IoTHubUri, deviceID, and deviceKey for the device that has been pre-created for you in the RM solution (because you likely are not an admin in that solution)
3. Edit the lines of code below that you already edited in Lab 2, to point to the new device and IoTHub associated with the RM pre-configured solution

private static string iotHubUri = "<iothubname>.azure-devices.net"; // <iothubname>.azure-devices.net

private string deviceId = "StudentXXDevice"; // replace XX with your studentID

private static string deviceKey = "<deviceKey>";

1. One of the things that a device in the RM pre-configured solution does is, on boot, send updated “DeviceInfo” JSON message, which contains meta-data about the device, as well as an array of “commands” supported by the device. For the lab, we will just build the JSON message in a string. In a real implementation, you would likely use a device object model that you could serialize to JSON. Add the following function to the StartupTask class in your app. Feel free to change the property values (manufacturer, modelno, serialno, platform, etc) to something fun for you ☺.

private void SendDeviceInfo()

{

string createdDateTime = DateTime.UtcNow.ToString("o");

// create a JSON string that represents the device metadata

string deviceInfo = "{\"DeviceProperties\":{\"DeviceID\":\"" +

deviceId + "\",\"HubEnabledState\":true,\"CreatedTime\":\"" +

createdDateTime + "\",\"DeviceState\":\"normal\",\"UpdatedTime\":null,\"Manufacturer\":\"Busbyland Electronics\",\"ModelNumber\":\"RPI35073\",\"SerialNumber\":\"35073\",\"FirmwareVersion\":\"1.0\",\"Platform\":\"Raspberry Pi 2\",\"Processor\":\"Intel Atom\",\"InstalledRAM\":\"4GB\",\"Latitude\":" +

latitude.ToString() + ",\"Longitude\":" + longitude.ToString() +

"},\"Commands\":[{\"Name\":\"ON\",\"Parameters\":null},{\"Name\":\"OFF\",\"Parameters\":null}],\"CommandHistory\":[],\"IsSimulatedDevice\":false,\"Version\":\"1.0\",\"ObjectType\":\"DeviceInfo\"}";

// send the metadata to IoTHub

SendDeviceToCloudMessagesAsync(deviceInfo);

}

1. To put the device on the map on the dashboard front page, we need to specify a latitude and longitude for your device. To keep all of the devices from just showing up overlaid over each other on the map, we want to make the locations unique. So, go to <http://www.bing.com/maps> and navigate to your hometown (or other significant spot to you), right click on the location on the map, and note the latitude and longitude. Add a couple of variables to the top of the StartupTask class and update with the lat and long you chose.

static double latitude = 33.788296;

static double longitude = -86.816925;

1. The final step is to add a call to the SendDeviceInfo() function in the Run() function. Add the following line of code in the Run() method above the call to SetupSerialConnection().Wait()

SendDeviceInfo();

1. Hit F5 to deploy and run the solution on the RPI device.
2. Confirm device integration to the Remote Monitoring solution

In this section, you will confirm that the device integration into the RM solution was successful.

1. Navigate back to the RM solution
2. On the Device tab, refresh the page and see that your device meta-data has been updated to what you specified in the previous section. Note that when you click on the device, you now have a new “Commands” link in the upper right section.
3. Click on the Commands link and we’ll test sending a command to the device. In the drop down list box, note you should have two commands available, ON and OFF. Select ON and click the Send button. Note after a few seconds that the LED on the Arduino device will come on. Also note the “pending” status of the command. That indicates that the command has been sent, but the RM solution has not yet received feedback as to the disposition of the command from the device. The code on the device ‘acknowledges’ the command, so a few refreshes of this page should update the command to ‘success’. Now send an OFF command to the device and ensure the LED goes off and after a refresh or two, the command status becomes ‘success’
4. Now navigate back to the dashboard and select your device, either by navigating to it on the map, or selecting it from the drop-down list box. Note the updated temp and humidity telemetry on the graph and the ‘meters’ at the bottom of the screen.
5. Play around and enjoy ☺