2017

Microsoft IoT Workshop Labs Guide

Ver:1.0

SHAW CHYN CHIA

@snakechia

Microsoft MVP : Windows Development

MCT

Introduction

The goal of this labs exercise is to familiarize you with some of the components and technologies associated with Internet of Thing (IoT). Along the way, you will experience deploying code, streaming sensor data to Microsoft Azure, aggregating data with Stream Analytics and reporting with Microsoft Power BI.

Hardware Needed:

- Development Computer
- Raspberry Pi 2 or 3 (highly recommended Raspberry Pi 3)
- Sense HAT for Raspberry Pi
- LEDs
- Tactile Button
- 220Ω Resistor
- Connection cable (male-female)
- Breadboard

Software Required:

- Computer with Microsoft Windows 10 Anniversary Update (update 1607 or build 14393.xxx)

- Microsoft Visual Studio 2015 Community Edition with Update 3 or above.
  "**Universal Windows App Development Tool**" MUST be selected.

- Windows 10 IoT Core Dashboard
  http://go.microsoft.com/fwlink/?LinkID=708576

- Device Explorer
  https://github.com/Azure/azure-iot-sdks/releases (Scroll down for SetupDeviceExplorer.msi)
- Windows IoT Remote Client
  (Windows Store Apps)

Others

- Microsoft Azure Account
  https://azure.microsoft.com/

- Microsoft Power BI access
  http://www.powerbi.com

Table of Contents

Lab 1: Prepare Windows 10 IoT Core for Raspberry Pi 2/3

Download & Install Windows 10 IoT Core for Raspberry Pi 2/3 into microSD card

1. Download and install **Windows 10 IoT Core Dashboard** from http://go.microsoft.com/fwlink/?LinkID=708576

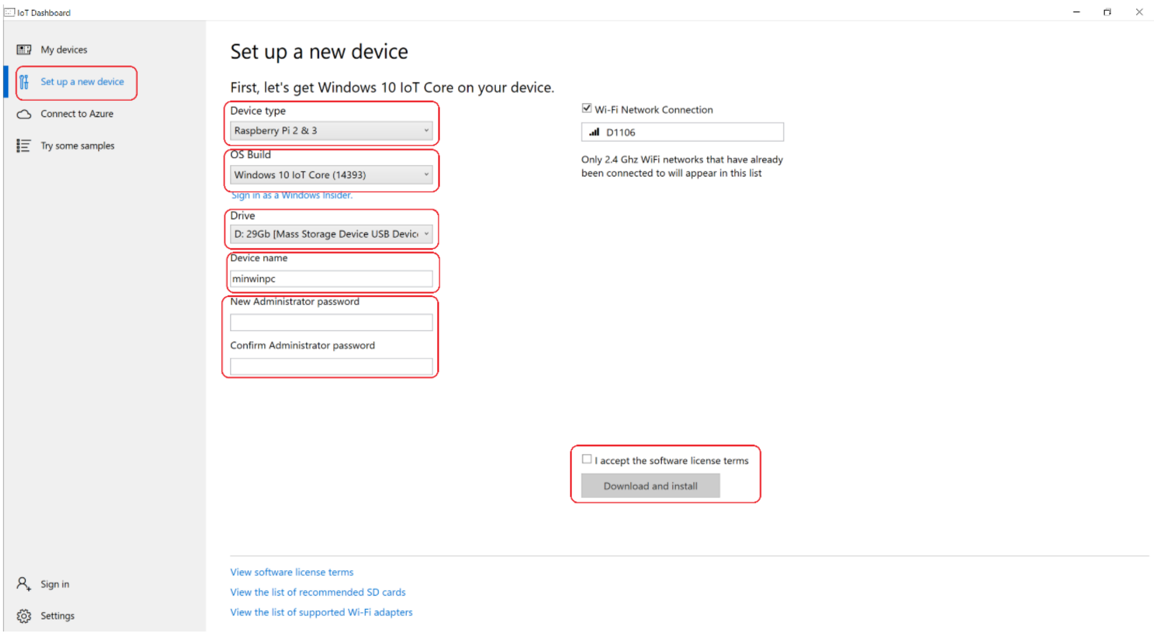
2. Press the Windows Key and type "Windows 10 IoT Core Dashboard" and "RUN".

3. Go to "**Setup up a new device**".

   - Make sure the "**Device type**" is set to "**Raspberry Pi 2 & 3**" and OS Build is not "**Windows Insider Preview**".

   - Modify your **Device Name** and type in your **New Password**.

   - Insert your microSD card via USB adapter to your PC, select correct drive letter at "**Drive**" section.



4. Check the "**I accept the software license terms**" to start the download and installation process.

   If this is the first time for the PC to perform this task, the PC will download Windows 10 IoT Core file into the PC, subsequent installation may just unpack the OS to your microSD card.

   This process may range from 5 – 20 minutes depend on the network condition. So, go and have a cup of tea or coffee.

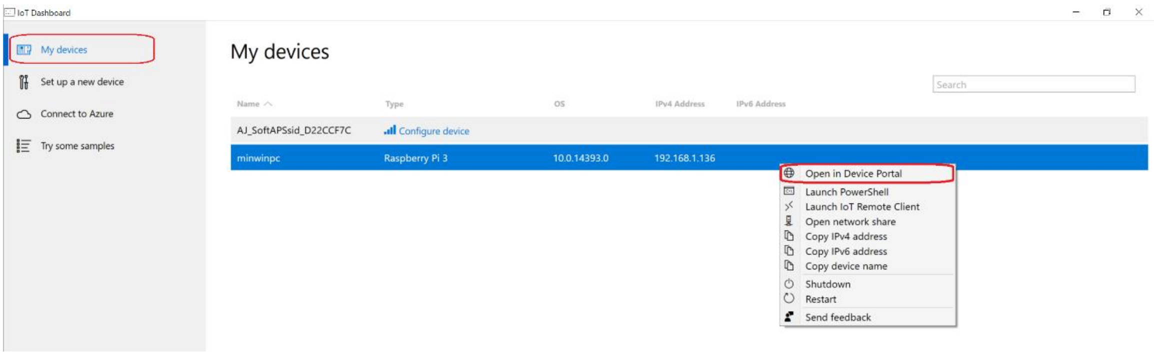
5. Once it done, you can remove the microSD card from the PC and insert it to Raspberry Pi 2/3

Lab 2: Connecting & Configuring Raspberry Pi 2/3

The Raspberry Pi should be connected to the development PC via a wired Ethernet connection. This connection is used for initial configuration or deployment and debugging if the device is not connected thru wireless.

1. Press the Windows Key and type “Windows 10 IoT Core Dashboard” and “RUN”.

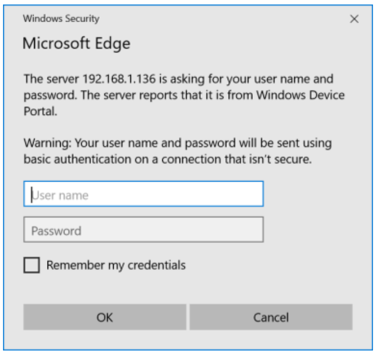
2. Go to “**My Devices**”, select your device, “**Right Click**” and click the “**Open in Device Portal**” icon.



If your device does not show up in the list, it is almost certainly because the network connection between your PC and the Raspberry Pi is having an issue.

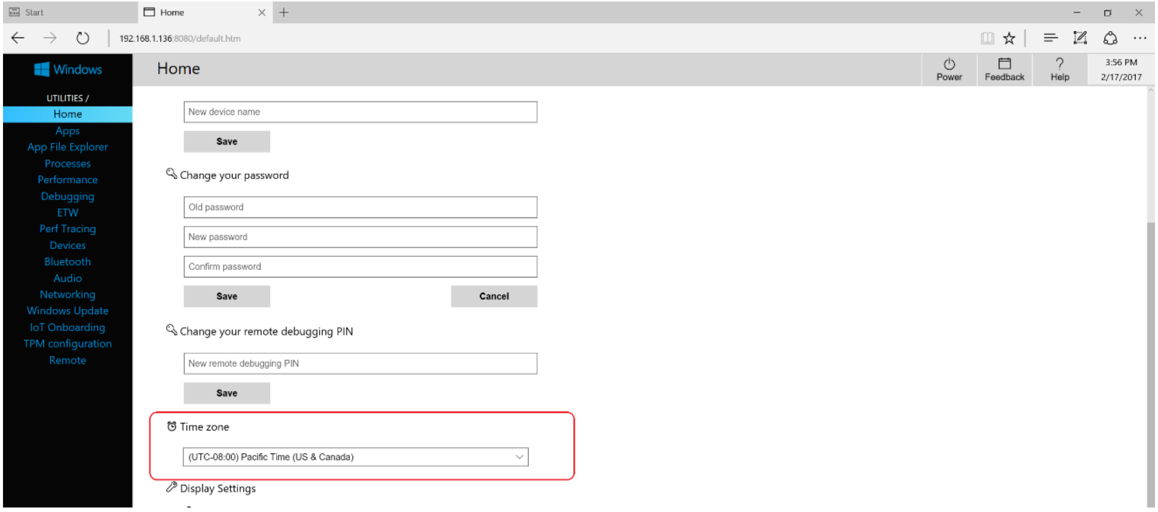
Alternatively, navigate to the default device url *http://minwinpc:8080*

3. Authenticate. The default credentials are Username: ***Administrator*** and Password: ***p@ssw0rd***, or your own password that set from previous Lab.

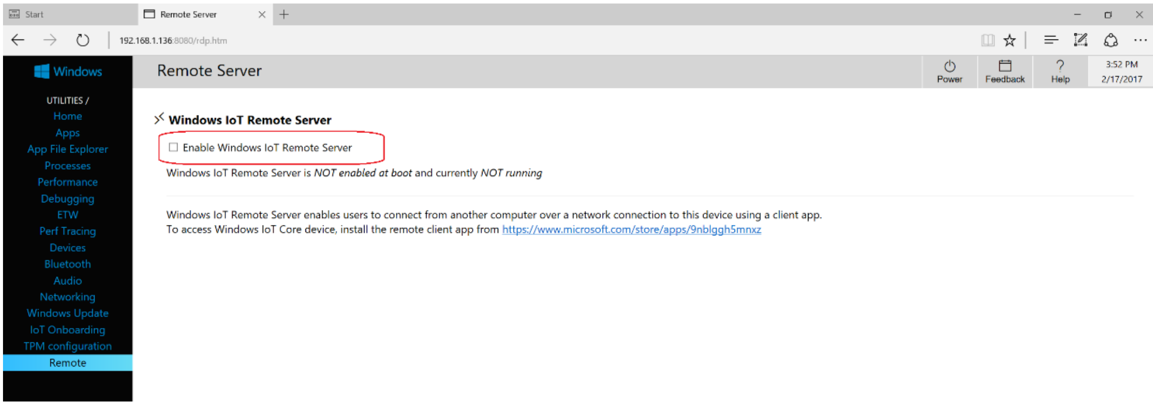


Windows Device Portal will launch and display the web management home screen.

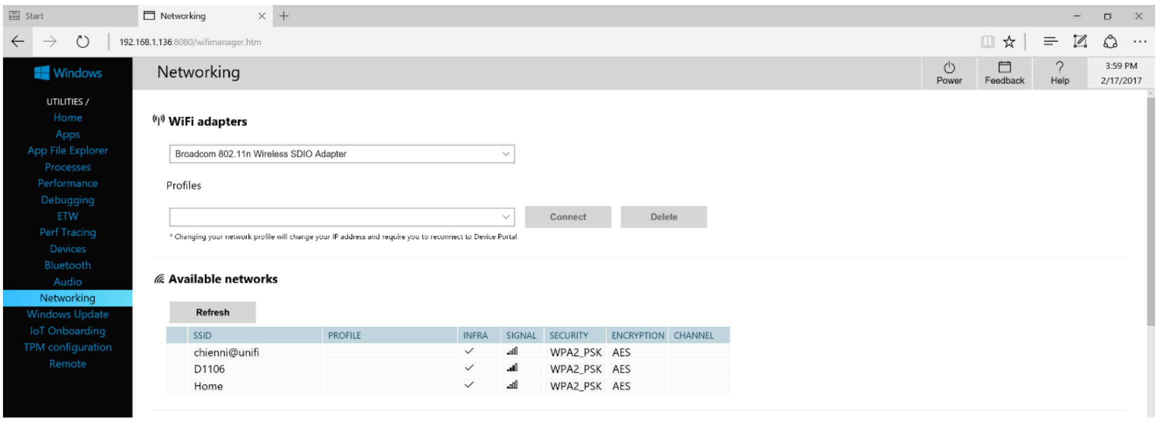
4. Verify Device Configuration
   - From the **Home** tab verify the time zone, date and time are correct.



   - From the **Remote** tab, verify that **Windows IoT Remote Server** is enabled. If it is not, then enable it.



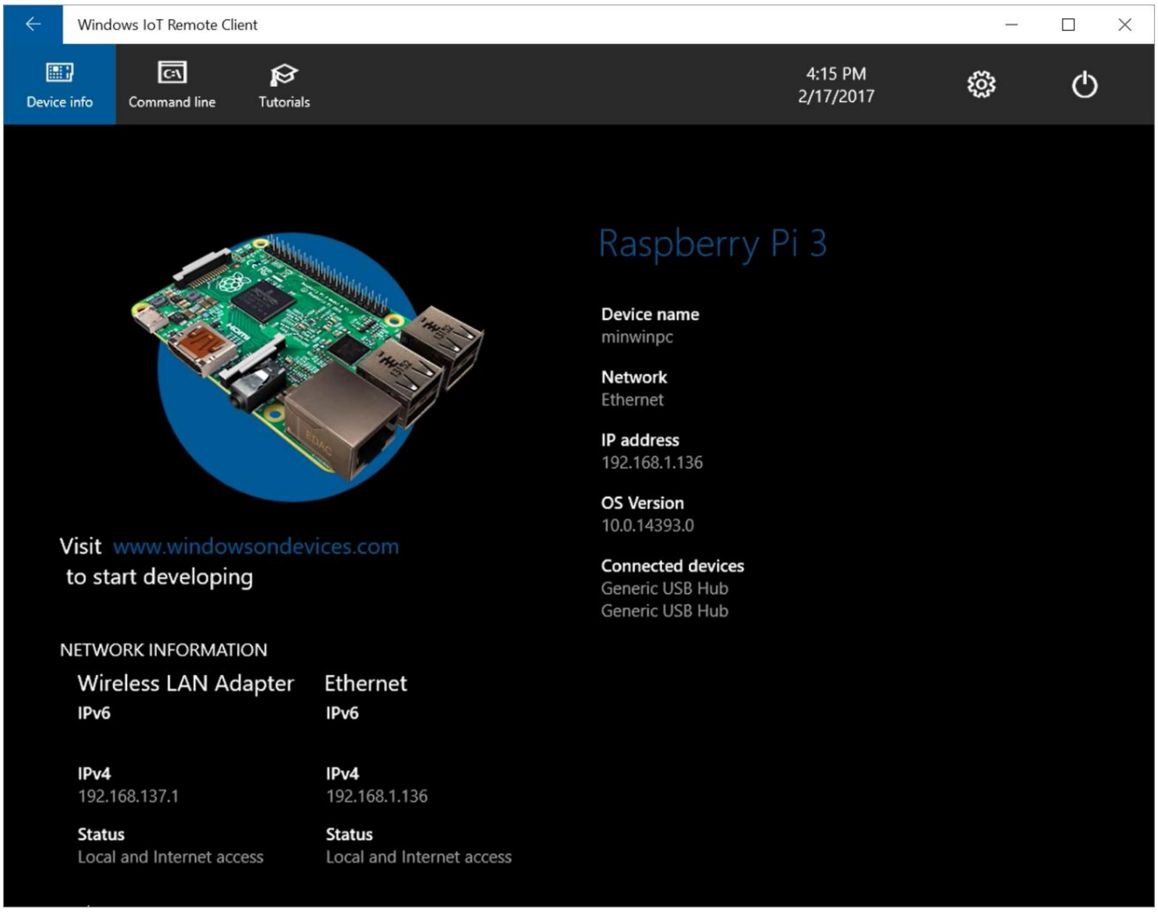
   - From the **Networking** tab, you can select the wireless network you would like to connect.



   - On the Windows Update tab, check for updates. It might take 30mins to download and another 30mins to install. Depend on your network connection.

5. Take a moment to explore the other abs in the Windows Device Portal.

6. Test Windows IoT Remote Client connection.
   Press the Windows key and type "**Windows IoT Remote Client**" and run.
   If "**Windows IoT Remote Client**" wasn't installed, go to Windows Store to search and install.



7. Select your device from the dropdown list or enter the IP Address of your device but not both.

   This will take a moment to connect. When it does, you will see the video output of the Raspberry Pi remoted to your desktop. Just like Remote Desktop features on your PC.

   Minimize the remote client application when you have verified that it is working.

Lab 3: Blink LED with Raspberry Pi 2/3

We'll create a simple LED blinking app and connect a LED to your Raspberry Pi 2/3 device.

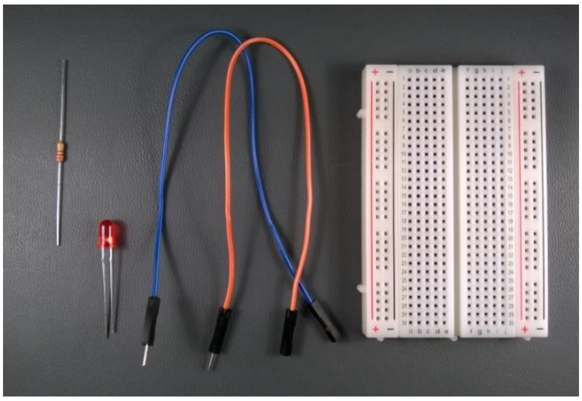
This is a headed UWP (Universal Windows Platform) sample app, and we will use this app to control the LED.

Be aware that the GPIO APIs are ONLY available on Windows 10 IoT Core, so this sample cannot run on your desktop or mobile device.

Connect the LED to your Raspberry Pi 2/3

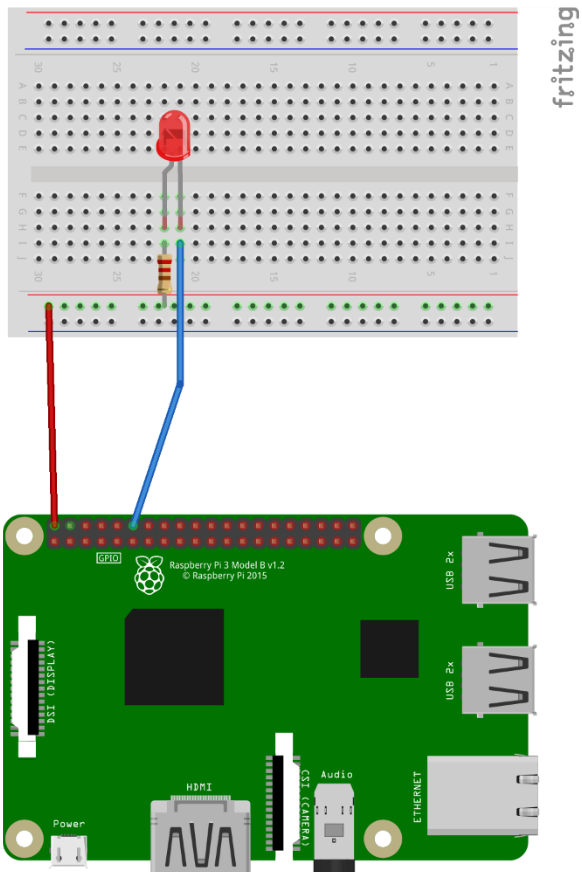
Components needed:

- A LED (any color you like)
- A 220Ω resistor
- A breadboard and couple of connector wires



Here is the circuit design:

1. Connect the shorter leg of the LED to GPIO 18 (pin 12 on the expansion header)
2. Connect the longer leg of the LED to the resistor.
3. Connect the other end of the resistor to one of the 5V pin on the Raspberry Pi 2/3. (here we use pin 2 on the expansion header)
4. Note that the polarity of the LED is important.

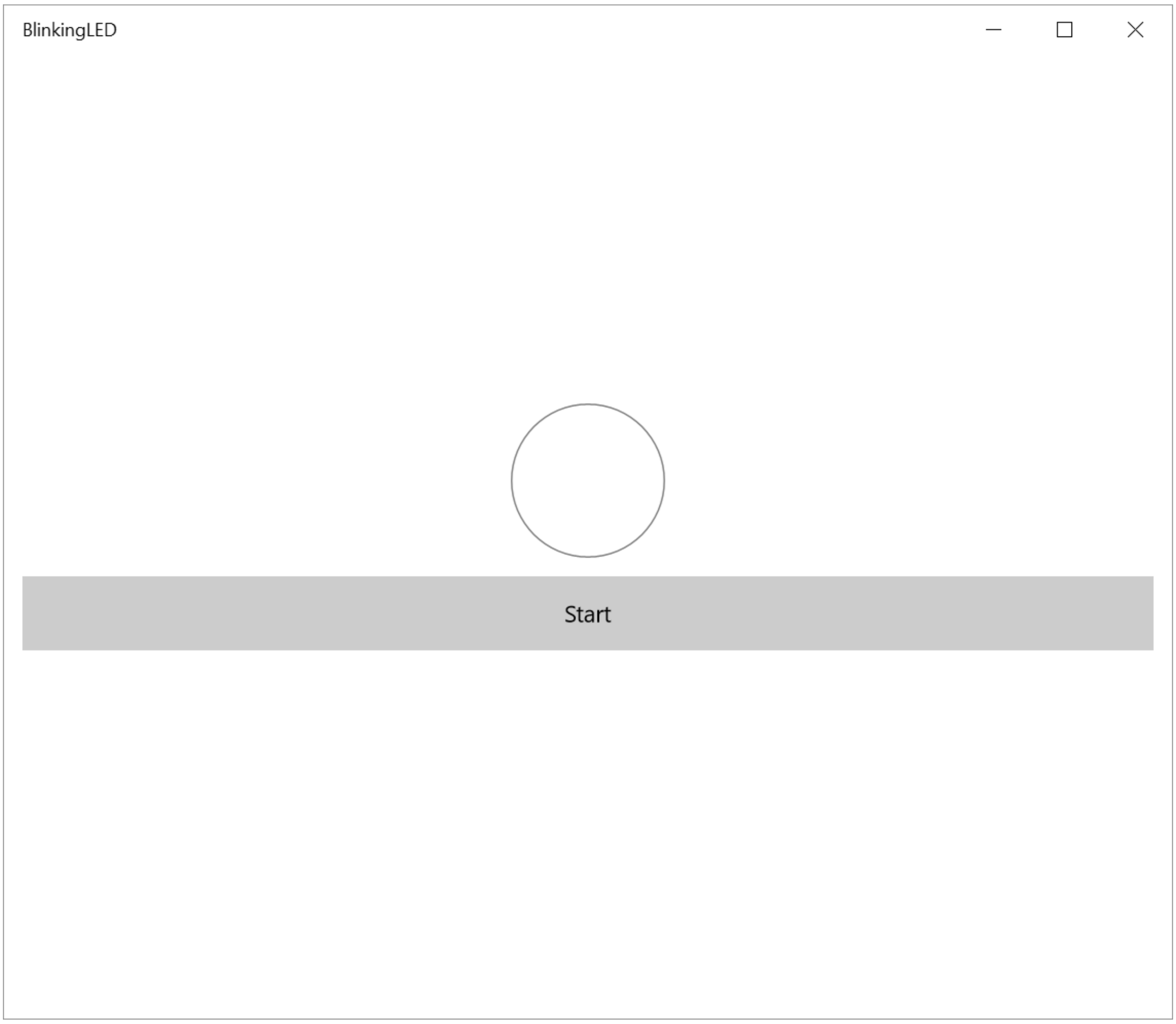


For the Raspberry Pi 2/3 GPIO Pins layout, please refer to Appendix section.

Building the blinking app

UI: MainPage.xaml

```
<StackPanel VerticalAlignment="Center">
        <Ellipse Name="led" Width="100" Height="100" Stroke="Gray" />
        <Button Name="btn" Content="Start" HorizontalAlignment="Stretch" Margin="12"
                Height="48" Click="btn_Click"/>
</StackPanel>
```



Code: MainPage.xaml.cs

This code is pretty simple, we going to use a button to start and stop the timer. By using the timer Tick event, we change the state of the LED as well as the UI to indicate the changes.

Define value

```
using Windows.Devices.Gpio;

DispatcherTimer timer;
GpioController gpio;
GpioPin ledpin;
GpioPinValue pinvalue;
```

Detect and Initialize GPIO Controller and GPIO Pin

```
public MainPage()
{
        gpio = GpioController.GetDefault();

        if (gpio == null)
        {
            btn.IsEnabled = false;
            return;
        }

        ledpin = gpio.OpenPin(18);
        ledpin.SetDriveMode(GpioPinDriveMode.Output);
```

- Leverage the new WinRT classes in the Windows.Devices.Gpio namespace to initialize the GPIO Controller.
- First, we use GpioController.GetDefault() to get the GPIO controller.
- If the device does not have a GPIO controller, this function will return null, and we will disable the “Start” button.
- We are going to open the pin #5 as per our circuit design by calling GpioController.OpenPin() with the pin value.
- We also set the ledpin to run in output mode using the GpioPin.SetDriveMode() function.

Define timer function

```
        timer = new DispatcherTimer();
        timer.Interval = TimeSpan.FromMilliseconds(300);
        timer.Tick += Timer_Tick;
}
```

```
private void Timer_Tick(object sender, object e)
{
    if (pinvalue == GpioPinValue.High)
    {
        pinvalue = GpioPinValue.Low;
        led.Fill = new SolidColorBrush(Colors.Transparent);
    }
    else
    {
        pinvalue = GpioPinValue.High;
        led.Fill = new SolidColorBrush(Colors.Red);
    }

    ledpin.Write(pinvalue);
}
```

- If the LED is On, turn it Off, else turn it On.
- By defining the GpioPin Value to “High” or “Low”, this will determine the current flow to LED hence, the LED will On and Off.
- Using GpioPin.Write() function to define the value of the GpioPin.
- Based on the LED On/Off status, we set the color of the circle too.

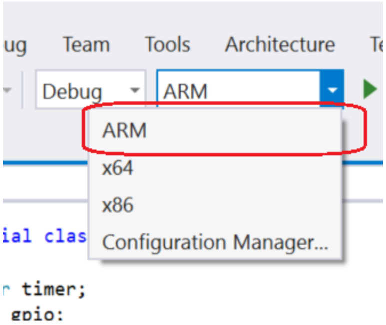
Start/Stop button

```
private void btn_Click(object sender, RoutedEventArgs e)
{
    if (btn.Content.ToString() == "Start")
    {
        btn.Content = “Stop”;
        timer.Start();
    }
    else
    {
        btn.Content = “Start”;
        timer.Stop();
    }
}
```

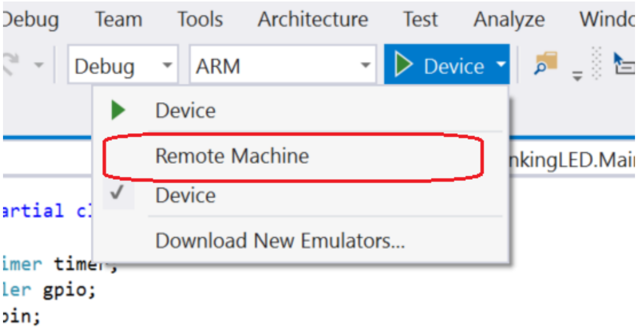
- Using this function to start/stop the timer.

Deploy your app

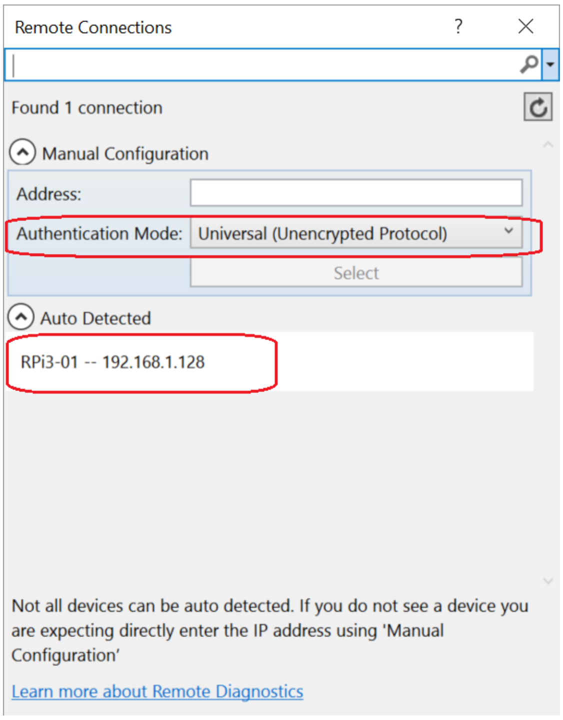
1. In Visual Studio, set the architecture in the toolbar dropdown to "**ARM**", this is because Raspberry Pi 2 or 3 are running on ARM architecture.



2. Next, in the Visual Studio toolbar, click on the "**Local Machine**" dropdown and select "**Remote Machine**".



3. Visual Studio will present the "**Remote Connection**" dialog. You can either enter your device name here or use the IP Address of your device which you can obtain from "**Windows 10 IoT Dashboard**".



4. Once entering the device name/IP address, select “**Universal**” for Windows Authentication, then click “**Select**”.

5. You can modify these values by navigating to the project properties.
   (Select **Properties** in the **Solution Explorer**) and choosing the “**Debug**” tab on the left.

6. Deploy the application to Raspberry Pi 2/3 by clicking the “Remote Device”.

Monitoring the Application

1. By using Windows IoT Remote Client, you should be seeing the Display output of the Raspberry Pi 2/3.

2. Use your mouse to click the “Start” button, and now you should be able to see the LED and the red dot on the screen is flashing.

Congratulation! You just completed your first Windows 10 IoT apps!

Lab 4: Push Button with Raspberry Pi 2/3

Base on previous lab, Blinking LED, we'll create a simple Push to turn On and Off the LED. Whenever the button being push, the LED will blink. The circle on the screen will display accordingly as it is a headed project.

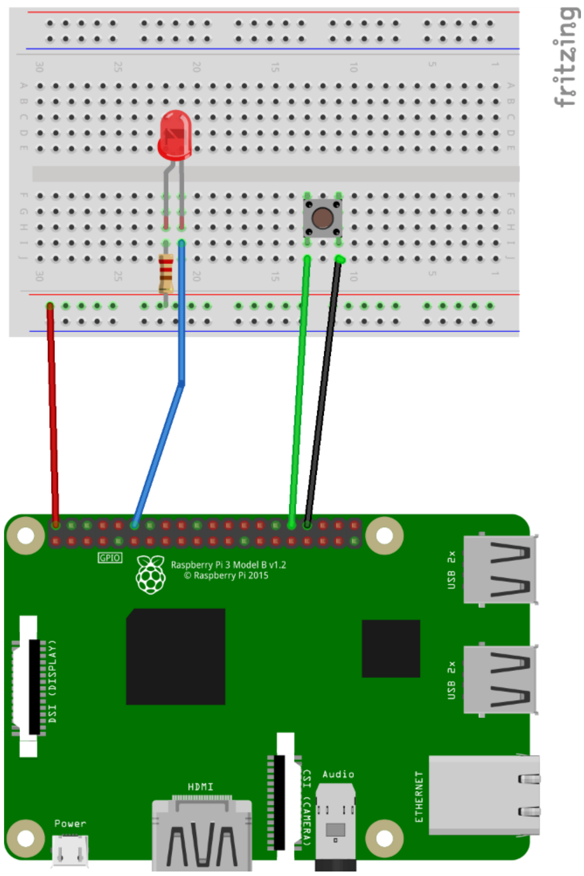
Be aware that the GPIO APIs are ONLY available on Windows 10 IoT Core, so this sample cannot run on your desktop or mobile device.

Components needed:

- A LED (any color you like)
- A 220Ω resistor
- A tactile button
- A breadboard and couple of connector wires

Here is the circuit design:

1. Connect the shorter leg of the LED to GPIO 18 (pin 12 on the expansion header)
2. Connect the longer leg of the LED to the resistor.
3. Connect the other end of the resistor to one of the 5V pin on the Raspberry Pi 2/3. (here we use pin 2 on the expansion header)
4. Note that the polarity of the LED is important.
5. Connect GPIO 12 (pin 32 on the expansion header) to one of the leg of tactile button.
6. Connect the other leg of the tactile button to Ground (pin 34 on the expansion header)

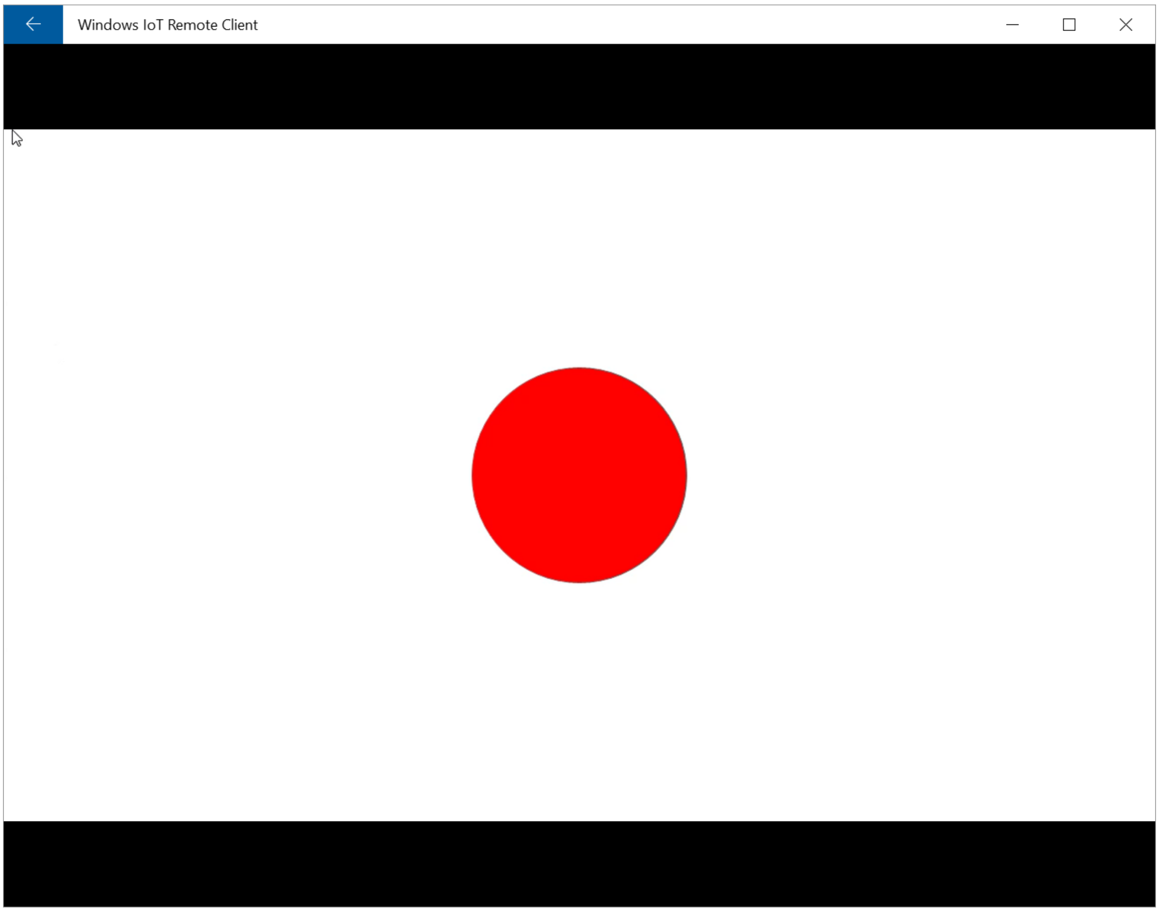


For the Raspberry Pi 2/3 GPIO Pins layout, please refer to Appendix section.

Building the Push Button app

UI: MainPage.xaml

```
<Ellipse Name="ellipse" Height="240" Width="240" Fill="Red" Stroke="Gray" />
```



Code: MainPage.xaml.cs

Based on the previous Lab 3 code, the Blinking LED, we maintain the circuit design yet adding in the new tactile button. As for the result, we will use the tactile button as a “Switch” to turn On and Off the LED.

Define value

```
using Windows.Devices.Gpio;

GpioController gpio;
GpioPin ledpin, pushbuttonpin;
GpioPinValue ledpinvalue;
```

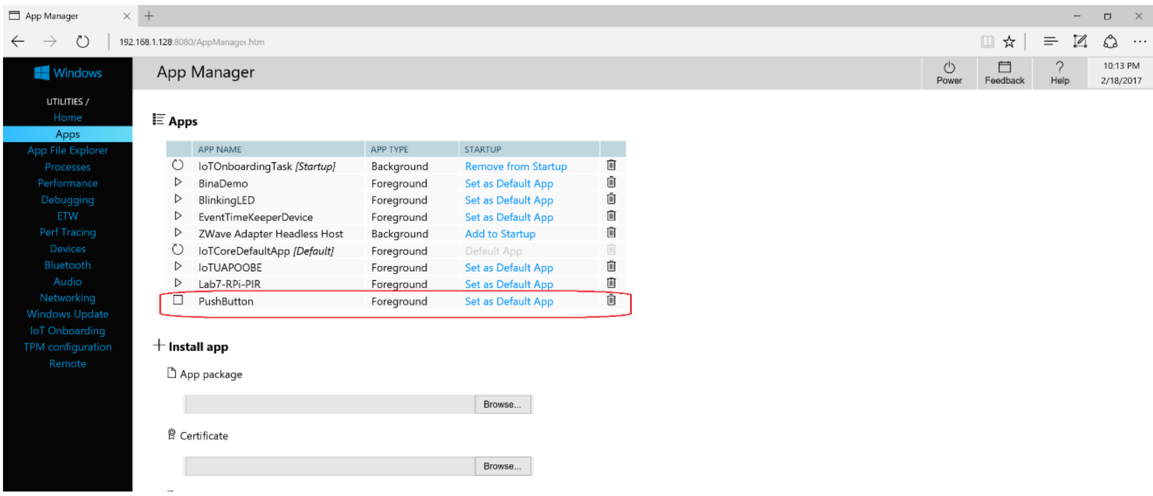
Detect and Initialize GPIO Controller and GPIO Pin

```
private void initGPIO()
{
    gpio = GpioController.GetDefault();

    if (gpio == null)
        return;

    ledpin = gpio.OpenPin(18);
    ledpin.SetDriveMode(GpioPinDriveMode.Output);
    ledpin.Write(GpioPinValue.Low);

    pushbuttonpin = gpio.OpenPin(12);

    // Check if input pull-up resistors are supported
    if (pushbuttonpin.IsDriveModeSupported(GpioPinDriveMode.InputPullUp))
        pushbuttonpin.SetDriveMode(GpioPinDriveMode.InputPullUp);
    else
        pushbuttonpin.SetDriveMode(GpioPinDriveMode.Input);

    // Set a debounce timeout to filter out switch bounce noise from a button press
    pushbuttonpin.DebounceTimeout = TimeSpan.FromMilliseconds(50);

    // Register for the ValueChanged event so when the push button being press,
    // function will be run.
    pushbuttonpin.ValueChanged += Pushbuttonpin_ValueChanged;

}
```

When Tactile Button being pressed

```
private async void Pushbuttonpin_ValueChanged(GpioPin sender,
GpioPinValueChangedEventArgs args)
{
    // toggle the state of the LED every time the button is pressed
    if (args.Edge == GpioPinEdge.FallingEdge)
    {
        ledpinvalue = (ledpinvalue == GpioPinValue.Low) ?
        GpioPinValue.High : GpioPinValue.Low;
        ledpin.Write(ledpinvalue);
    }

    // need to invoke UI updates on the UI thread because this event
    // handler gets invoked on a separate thread.
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
    {
        if (args.Edge == GpioPinEdge.FallingEdge)
        {
            ellipse.Fill = (ledpinvalue == GpioPinValue.Low) ?
            new SolidColorBrush(Colors.Red) : new SolidColorBrush(Colors.Transparent);
        }
    });
}
```

- Follow Lab 3 Deploy App to Raspberry Pi steps to deploy this app to your device.
- Using Windows IoT Remote Client to monitor the Screen output.
- You can also start the app thru the Device Portal by navigate to "Apps" section and look for the App Name that you created. Please the "Play" button to start your app.



Congratulation! You just completed your second Windows 10 IoT apps!

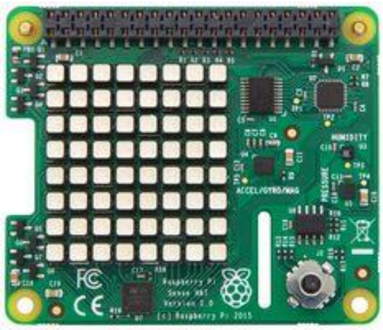
Lab 5: Mini Weather Station with Sense HAT and Raspberry Pi 2/3

Sense HAT is attached on top of the Raspberry Pi via the 40 GPIO pins. The Sense HAT has several integrated circuit based sensors, such as: Gyroscope, Accelerometer, Magnetometer, Barometer, Temperature, Humidity sensors, it also equipped with 8x8 LED matrix display and a small 5 button joystick.

In this Lab, we are going to read the Temperature, Humidity sensors values, display on the 8x8 LED matrix as well as output to screen if the device is connected to a screen.

Components needed:

- Sense HAT



What we going to achieve in this Lab:

1. Build a Sensor Model for our Mini Weather Station.
2. Read values from Sense HAT and put into our sensor model.
3. Display sensor data on the device screen output.
4. Display sensor data on Sense HAT 8x8 LED matrix.

Building the Weather Station

Before we can build our Weather Station, some ground work need to be done. Let's install RPi.SenseHat and Json.NET library from nuget

1. Json.Net

```
PM> Install-Package Newtonsoft.Json
```

2. RPi.SenseHat

```
PM> Install-Package Emmellsoft.IoT.RPi.SenseHat
```

Construct Sensor Data Model

As we know that, Sense HAT for Raspberry Pi is going to return us various of sensors data, our goal is just to have Temperature, Pressure and Humidity data, hence our data model will build around it.

```
public class SensorData
{
    public double temperature { get; set; }
    public double pressure { get; set; }
    public double humidity { get; set; }
    public DateTime createdAt { get; set; }
}
```

Build our own StringFormatConverter

The current version of XAML for UWP lack of **StringFormatConverter**, we need to write our own code for it. This is pretty much simple. The future release of UWP will have built in **StringFormatConverter**. The StringFormatConverter code as follow:

```
public class StringFormatConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
             string language)
    {
        if (value == null)
            return null;

        if (parameter == null)
            return value;

        return string.Format((string)parameter, value);
    }

    public object ConvertBack(object value, Type targetType, object parameter,
             string language)
    {
        throw new NotImplementedException();
    }
}
```

Building the UI

1. Open MainPage.xaml.

2. Insert the following code before

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
```

```
<Page.Resources>
    <local:StringFormatConverter x:Key="StringFormatConverter" />
</Page.Resources>
```

3. Replace

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
```

with

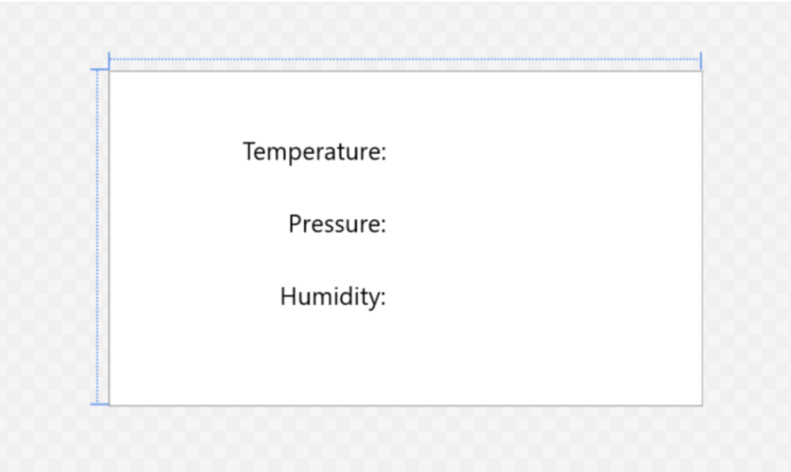
```
<Grid Name="dataGrid" Background="{ThemeResource
ApplicationPageBackgroundThemeBrush}">
```

4. Insert the following code between

```
<StackPanel VerticalAlignment="Center">
    <TextBlock Text="Mini Weather Station" FontSize="24" HorizontalAlignment="Center"
               Margin="24" />
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <TextBlock Text="Temperature:" HorizontalAlignment="Right" Margin="12" />
        <TextBlock Text="{Binding temperature,
               Converter={StaticResource StringFormatConverter},
               ConverterParameter='{}{0:0.00 C}'}" Grid.Column="1"
               Margin="12" FontWeight="Thin" />
    </Grid>

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <TextBlock Text="Pressure:" HorizontalAlignment="Right" Margin="12" />
        <TextBlock Text="{Binding pressure,
               Converter={StaticResource StringFormatConverter},
               ConverterParameter='{}{0:0.000 Pa}'}" Grid.Column="1"
               Margin="12" FontWeight="Thin" />
    </Grid>

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <TextBlock Text="Humidity:" HorizontalAlignment="Right" Margin="12" />
        <TextBlock Text="{Binding humidity,
            Converter={StaticResource StringFormatConverter},
            ConverterParameter='{}{0:0}'}" Grid.Column="1" Margin="12"
            FontWeight="Thin" />
    </Grid>
```

```
    <TextBlock Text="{Binding createdAt,
        Converter={StaticResource StringFormatConverter},
        ConverterParameter='{}{0:HH:mm:ss}'}" HorizontalAlignment="Center"
        Margin="24,24,24,6" />
    <TextBlock Text="{Binding createdAt,
        Converter={StaticResource StringFormatConverter},
        ConverterParameter='{}{0:MMM dd, yyyy}'}" HorizontalAlignment="Center"
        Margin="24,6,24,24" />
</StackPanel>
```

This is how the UI look like in Visual Studio



Writing the Code in MainPage.xaml.cs

1. Open MainPage.xaml.cs

2. Define the following objects.

```
using Emmellsoft.IoT.Rpi.SenseHat;

ISenseHat senseHat;
SensorData sensordata = new SensorData();
DispatcherTimer timer;
```

3. Put the following code into `public MainPage()`

```
timer = new DispatcherTimer();
timer.Interval = TimeSpan.FromSeconds(1);
timer.Tick += Timer_Tick;
```

4. To initialize Sense HAT, we have to use “Try Catch method” to prevent the app crash if there is no Sense HAT installed. Insert the following code

```
protected override async void OnNavigatedTo(NavigationEventArgs e)
{
    try
    {
        senseHat = await SenseHatFactory.GetSenseHat();
        timer.Start();
    }
    catch
    {
        return;
    }
}
```

5. Define the timer Tick event.

```
private void Timer_Tick(object sender, object e)
{
    getsensordata();
}
```

6. Read various sensors data and display it to device screen output.

```
private void getsensordata()
{
    senseHat.Sensors.HumiditySensor.Update();
    senseHat.Sensors.PressureSensor.Update();

    sensordata.temperature = (double)senseHat.Sensors.Temperature;
    sensordata.pressure = (double)senseHat.Sensors.Pressure;
    sensordata.humidity = (double)senseHat.Sensors.Humidity;
    sensordata.createdAt = DateTime.UtcNow;

    // this simple app we do not implement NotifyPropertyChanged
    // hence we manually refreshed the databinding.
    dataGrid.DataContext = null;
    dataGrid.DataContext = sensordata;
}
```

7. You may deploy the application to your Raspberry Pi 2/3 and use **Windows IoT Remote Client** to see the output screen.

8. To be able to display the information on the 8x8 LED matrix on Sense HAT, we need to define the following:

```
using Emmellsoft.IoT.Rpi.SenseHat.Fonts.SingleColor;

ISenseHatDisplay display;
TinyFont tinyFont = new TinyFont();
int count = 0;
```

9. The following function is to display temperature and humidity value from Sense HAT sensors, with 5 second interval.

```
private void displayDataOn8x8()
{
    double value = sensordata.temperature;
    Color color = Colors.Blue;

    switch (count)
    {
        case 1:
            value = double.Parse(senseHat.Sensors.Humidity.ToString());
            color = Colors.Red;
            break;

        default:
            value = sensordata.temperature;
            color = Colors.Blue;
            break;
    }

    display = senseHat.Display;
    display.Clear();
    tinyFont.Write(display, ((int)Math.Round(value)).ToString(), color);
    display.Update();

    count++;

    // Reset counter to 0 so that the LED matrix can show temperature value
    if (count > 1)
    {
        count = 0;
    }
}
```

10. Call `displayDataOn8x8()` function from `getsensordata()` by adding `displayDataOn8x8()` to the last line of `getsensordata()` function.

11. You may now re-deploy the application to your Raspberry Pi 2/3 and you should see the LED matrix will start display value and switch every 5 seconds.

Congratulation! You just completed building a Mini Weather Station with Sense HAT!

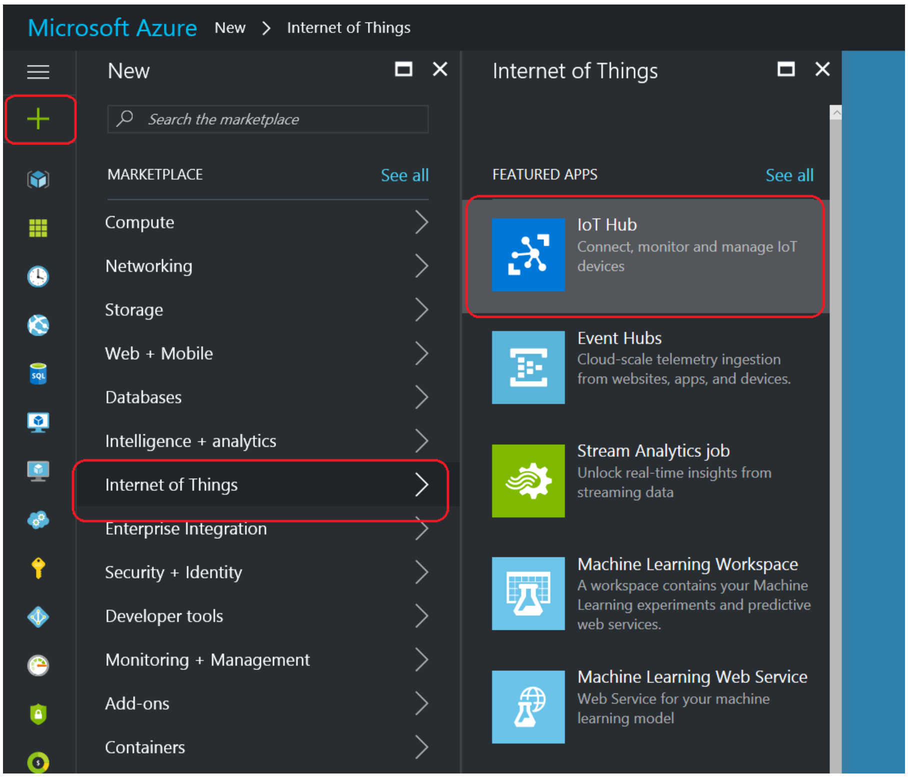
Lab 6: Create Azure IoT Hub

This lab assumes that you have either been provided with or created your own Azure account.

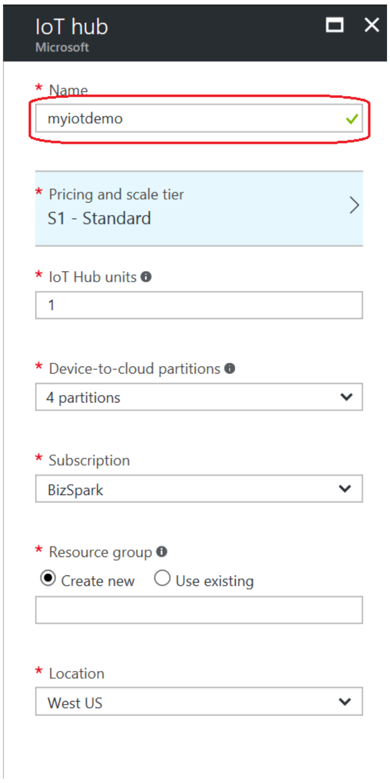
You need to create an IoT Hub for your device to connect to. The following steps show you how to complete this task using the Azure Portal.

1. Sign into the Azure Portal, http://portal.azure.com

2. In the Jumbar, click **NEW**, then click **Internet of Things**, and follow by **Azure IoT Hub**.



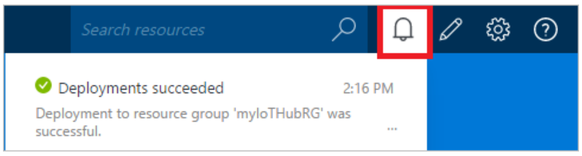
3. In the IoT Hub blade, choose the configuration for your IoT hub.



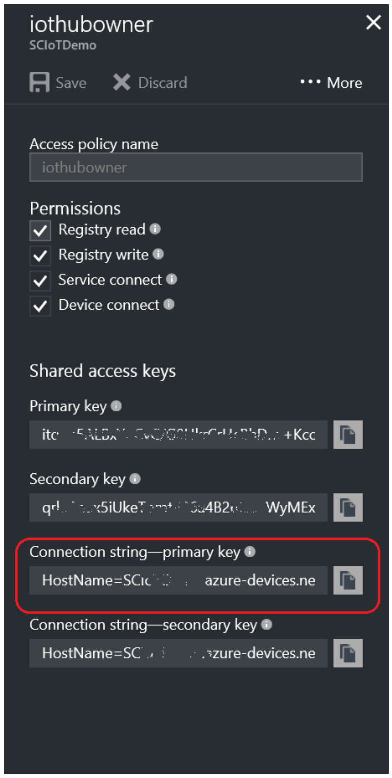
- In the **Name** box, enter a name for your IoT Hub. If the **Name** is valid and available, a **green** check mark appears in the **Name** box.
- Select a **Pricing and scale tier**. This workshop does not require a specific tier.
- In **Resource group**, create a new resource group, or select an existing one. For more information about Using Resource Groups, please visit https://azure.microsoft.com/en-us/documentation/articles/resource-group-portal/
- In **Location**, select the location to host your IoT Hub.

4. When you have chosen your IoT Hub configuration options, click Create. It can take a few minutes for Azure to create your IoT Hub. To check the status, you can monitor the progress on the Startboard or in the Notifications panel.

5. When the IoT hub has been created successfully, open the blade of the new IoT Hub, make a note of the **Hostname**, and then click the **Key** icon.



6. Click the **iothubowner** policy, then copy and make note of the **connection string** in the **iothubowner** blade.



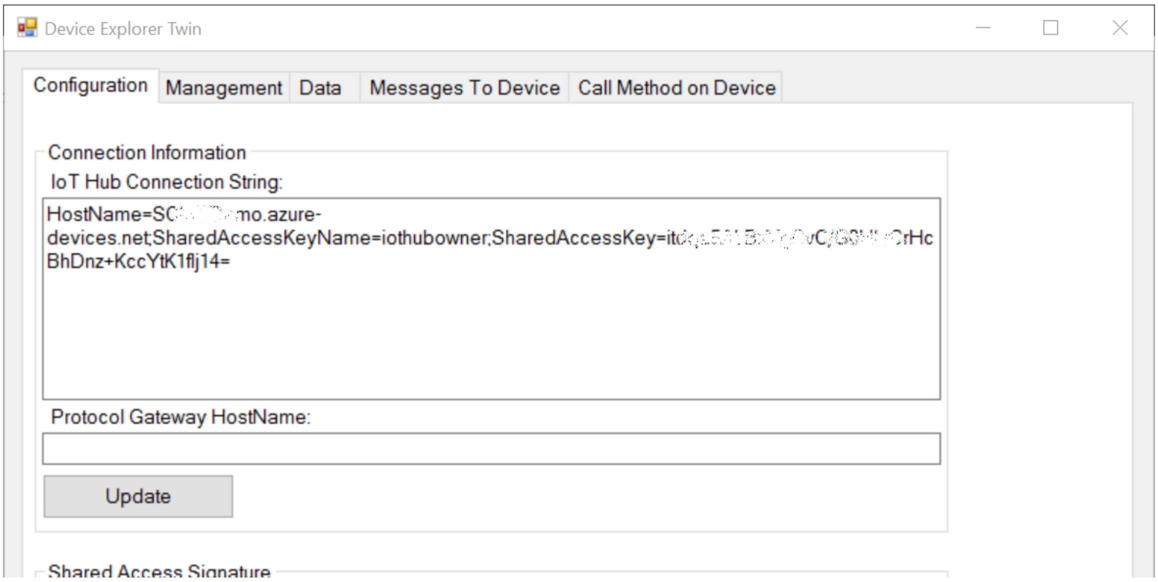
You have now created your IoT Hub and have the hostname and connection string you needed to complete the rest of this workshop.

Lab 7: Registering Your Device with Azure IoT Hub

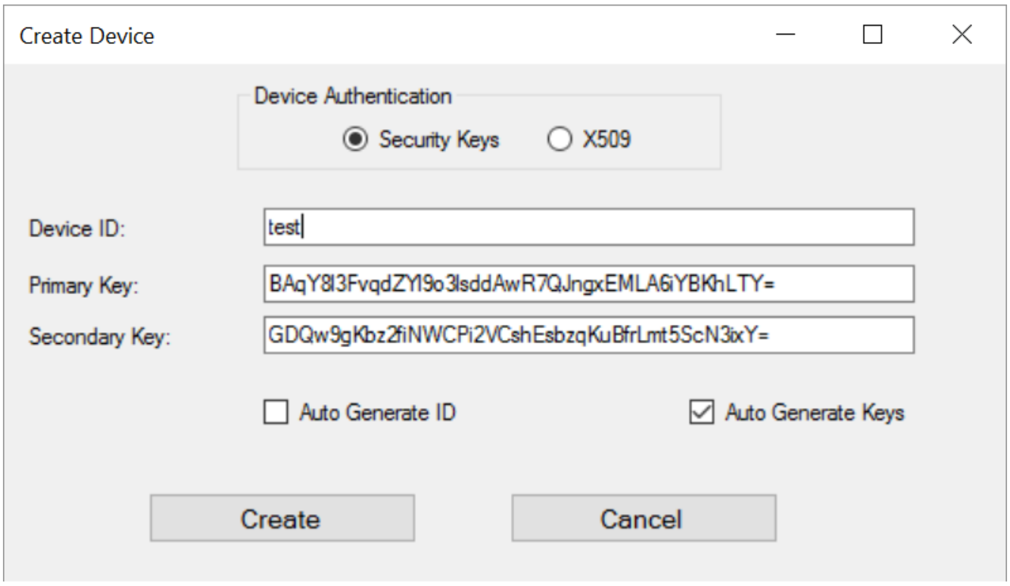
You must register your device in order to be able to send and receive information from the Azure IoT Hub. This is done by registering a Device Identity in the IoT Hub.

1. Press the Windows key and type “**Device Explorer**” and run the app.
   If “Device Explorer” is not installed, then install it from https://github.com/Azure/azure-iot-sdks/releases (Scroll down for SetupDeviceExplorer.msi)

2. Paste the **IoT Hub Connection String** provided by the previous Lab exercise into the **IoT Hub Connection String** field and click **Update**.

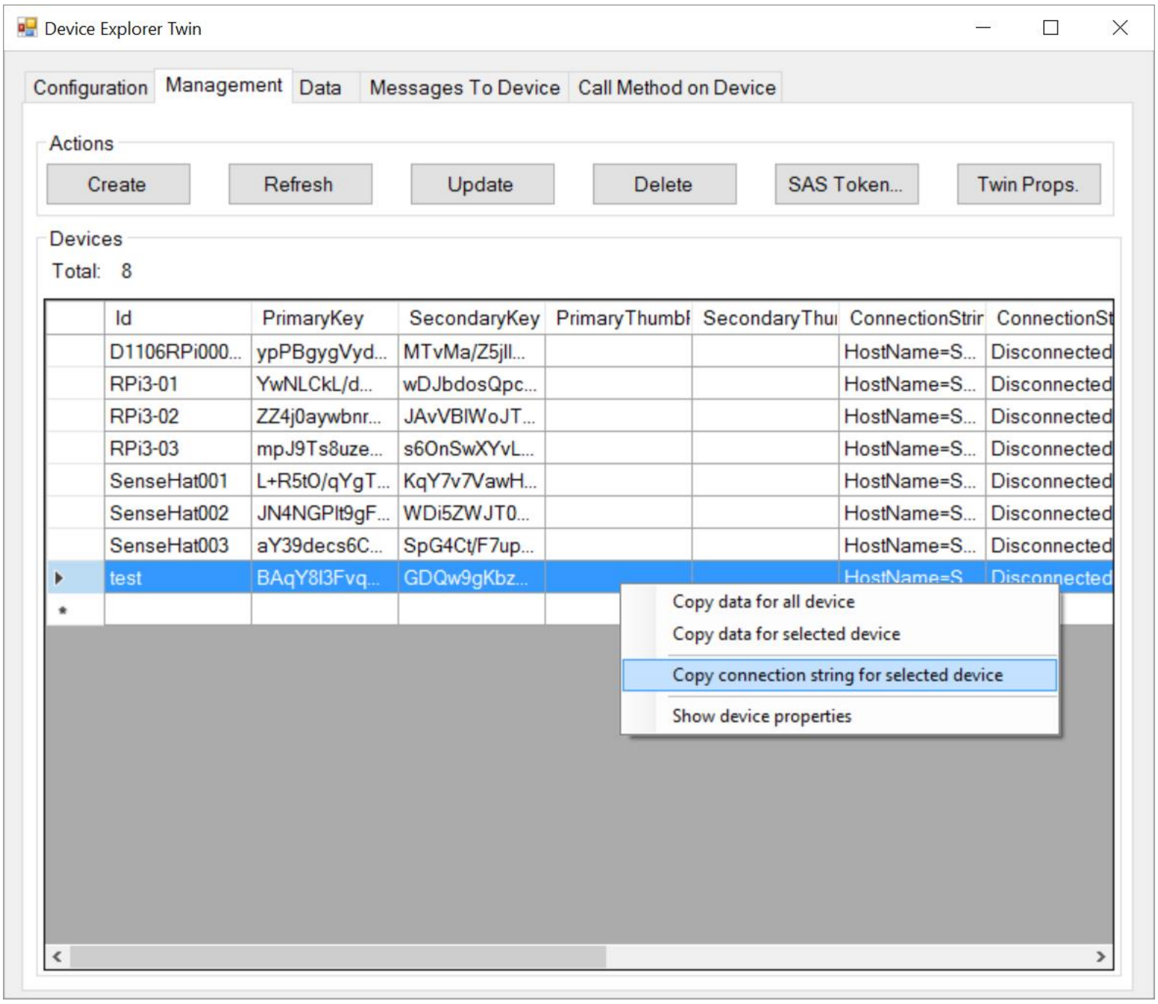


3. Go to the **Management** tab and click on the **Create** button. The Create Device popup will be displayed. Fill the **Device ID** field with a new ID for your device. For example: RPi3, then click on **Create**.



4. Once the device identity is created, it will be displayed in the grid. Right click on the identity you just created, select **Copy Connection String for selected device**, the connection string will be copied to the clipboard.

   This unique connection string allows a device to authenticate and communicate securely with Azure IoT Hub.



Lab 8: Update Mini Weather Station to send/receive data to Azure IoT Hub

Based on Lab 5: Mini Weather Station, we are going to send the temperature, pressure and humidity values to Azure IoT Hub by using the credential we created in Lab 6 and 7.

In this Lab, we are going to achieve the following:

1. Modified the **Lab 5** codes and send the sensordata to **Azure IoT Hub**.
2. Install Azure SDK
3. Verified the communication with “**Device Explorer**”.
4. Receive data sent from Azure IoT Hub (via “Device Explorer”)

Send Data to Azure IoT Hub

1. Install Azure Device SDK from nugget command line
   PM> Install-Package Microsoft.Azure.Devices.Client

2. Define the following:
   You can obtain the *deviceconnectionstring* from “**Device Explorer**” by highlight your targeted device and “**Right Click**”, select “Copy connection string for selected device”.

```
DeviceClient deviceClient;
string deviceName = "<Your device name>";
string deviceconnectionstring = "<Your device connection string>";
```

3. Put the following code into `public MainPage()` to initial the connection for your device to Azure IoT Hub.

```
deviceClient = DeviceClient.CreateFromConnectionString(deviceconnectionstring);
```

4. Create a function to send the data to Azure IoT Hub.

```
private async Task sendDataToAzureIoTHub(string message)
{
    try
    {
        var msg = new Message(Encoding.UTF8.GetBytes(message));
        await deviceClient.SendEventAsync(msg);
    }
    catch
    { }
}
```

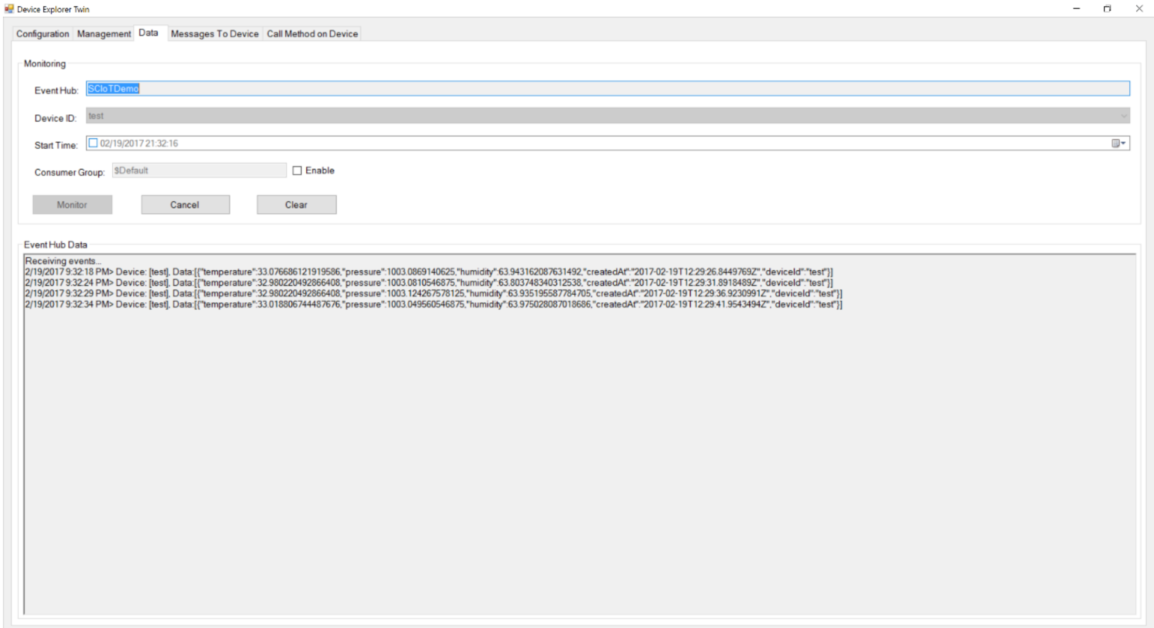
5. Finally, we need to convert the **sensordata** to a `string`, so that it can be send to Azure IoT Hub. Insert the following code into end of the `getsensordata()` section. It will send the data to cloud every 5 seconds.

```
sensordata.deviceId = deviceName;
string message = JsonConvert.SerializeObject(sensordata);
sendDataToAzureIoTHub(message);
```

6. Deploy the application to your device like previous labs.

7. Press the Windows key and type "**Device Explorer**" and run the app.
   Select the "**Data**" tab, follow by choose your device name at "**Device ID**" section and click "**Monitor**" button.

8. You should be able to see the data flowing in.



Received Message from Azure IoT Hub

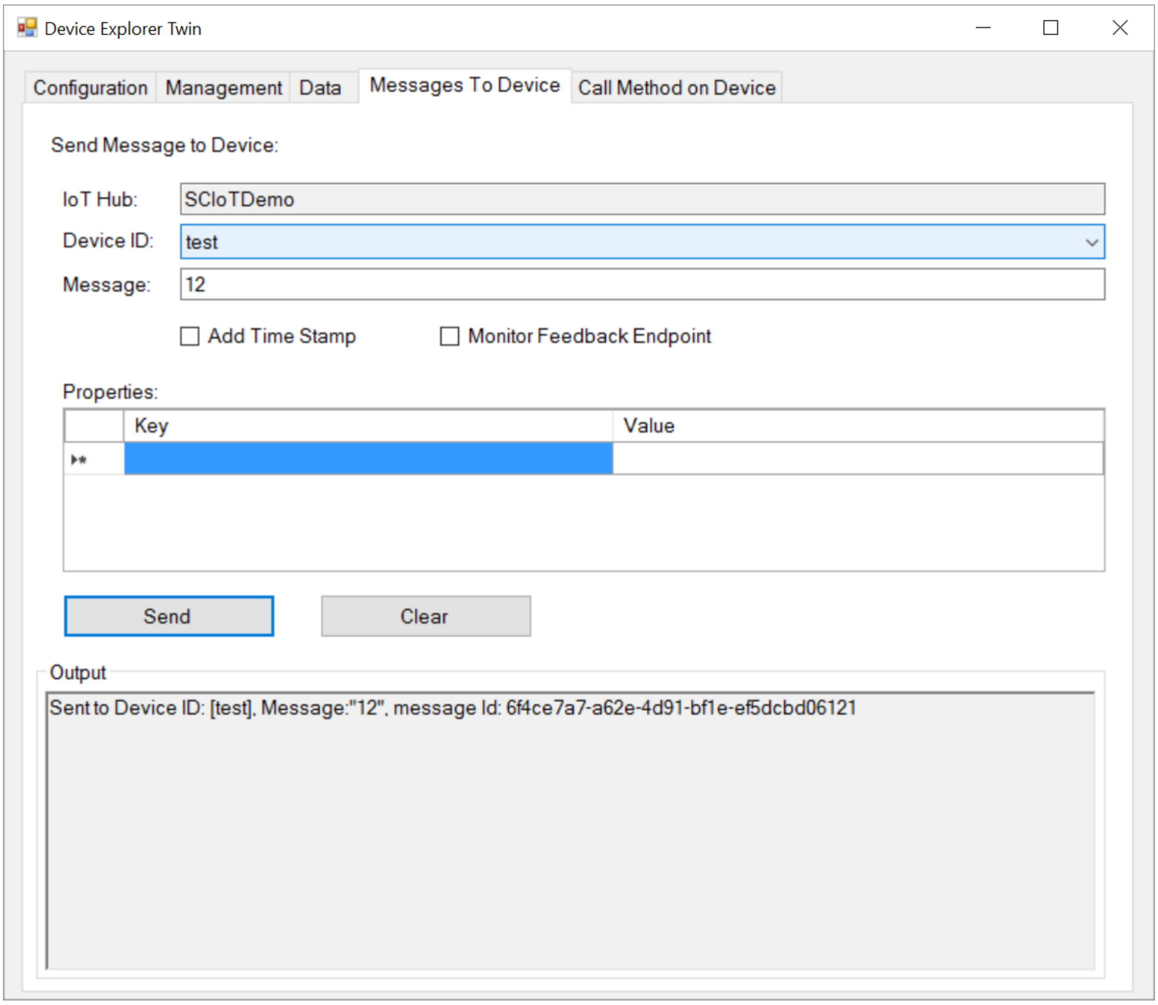
9. Next, we are going to add the codes that enable the device to receive data from Azure IoT Hub. To achieve this task, we are going to add the following function:

```
public async Task ReceiveDataFromAzureIoTHub()
{
    try
    {
        Message receivedMessage;
        string messageData;

        while (true)
        {
            receivedMessage = await deviceClient.ReceiveAsync();
            if (receivedMessage != null)
            {
                messageData = Encoding.ASCII.GetString(receivedMessage.GetBytes());
                await deviceClient.CompleteAsync(receivedMessage);

                if (messageData.Length > 2)
                    return;

                timer.Stop();

                display.Clear();
                tinyFont.Write(display, messageData, Colors.Green);
                display.Update();

                timer.Start();
            }
        }
    }
    catch
    { }
}
```

10. The timer will stop for the 8x8 LED matrix to display incoming message, after that it will be restarted. As for the result, the incoming message will be display for about 5 second before it being updated by temperature and humidity info.

11. Due to the TinyFont, it can't display the message longer than 2 characters, as for the result, we ignore any message that longer than 2 characters.

12. Navigate to “Message To Device” tab in “Device Explorer”.
Enter the your message in the Message Box, and press “Send”. Once the message been sent, you will receive the message ID.



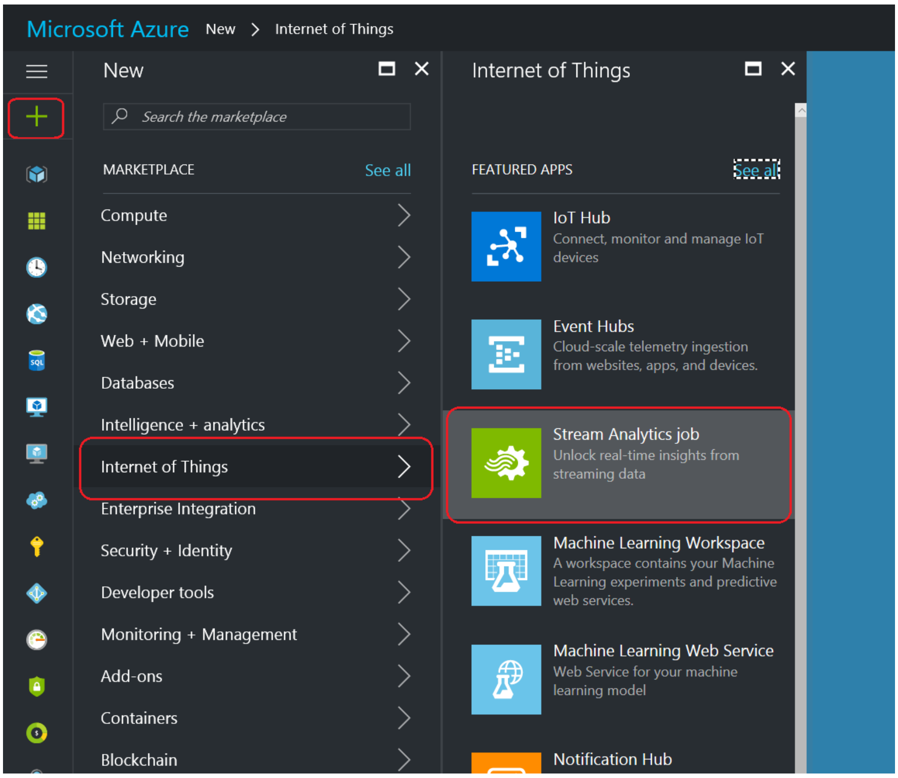
Congratulation! You just make your weather station an IoT device. :)

Lab 9: Create A Stream Analytics Job

Before the information can be delivered to Power BI, it must be processed by a Stream Analytics Job. To do so, an Input, Output and Query for the job must be defined. As the Raspberry Pi devices are sending information to an IoT Hub, it will be set as the input for the job, and Power BI as the output.

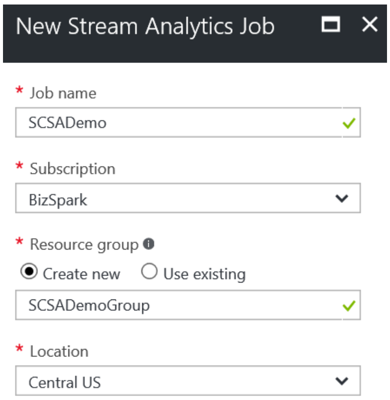
1. Sign into the Azure Portal, http://portal.azure.com

2. In the Jumbar, click **NEW**, then click **Internet of Things**, and follow by **Stream Analytics Job**.

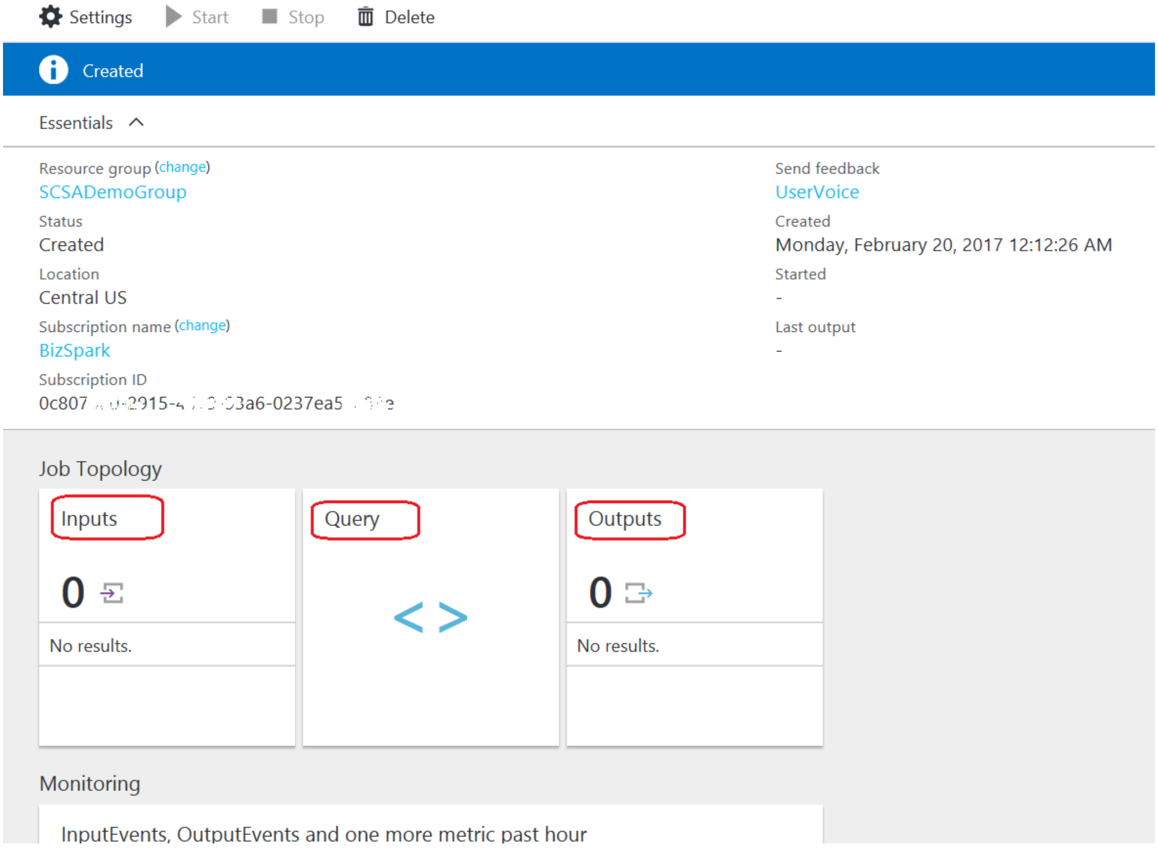


3. Define the Stream Analytics Job then click Create. It will take approximately 30 – 45 seconds to create the job.

4. Open the newly created Stream Analytics Job configuration blade.



5. Familiarize yourself with the Stream Analytics configuration blade. Note the Input, Query and Output zones.



As you can see, the Start Button is disabled since the job is not configured yet.

Stream Analytics – Configure New Input

An Input defines the data source for the Stream Analytic job. As in this workshop, it is the input from your Raspberry devices.

1. From the Stream Analytics Configuration blade, select
   Input -> Add -> the specify parameters -> Create -> Close the Input blade.

Settings for New Input Blade

Input alias
SenseHatTelemetryHub

Source
IoT Hub

Subscription
Provide IoT Hub settings manually

IoT Hub
Provide in previous Lab exercise

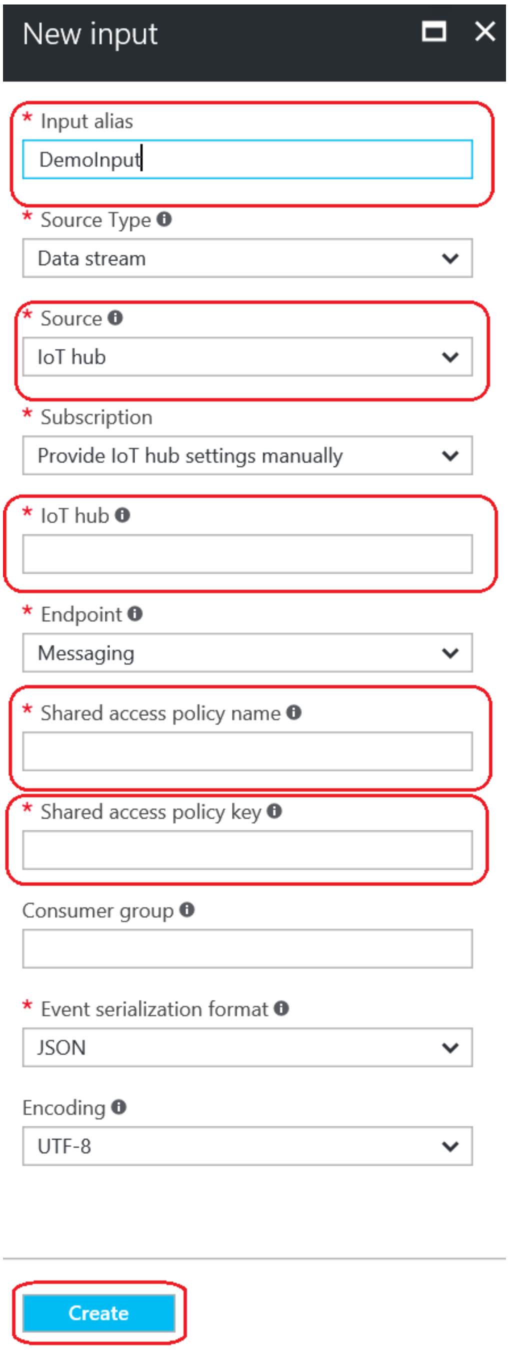
Shared access policy name
iothubowner

Shared access policy key
Provide in previous Lab exercise

Consumer group
Provide in previous Lab exercise

Click Create
It will take a moment or two to create the Input stream.

Close the Input Blade



Stream Analytics – Configure New Output

An Output defines the output destination for the Stream Analytics job. As in this workshop, the Output will be Power BI.

1. From the Stream Analytics Configuration blade, select
   Output -> Add -> the specify parameters -> Create -> Close the Output blade.

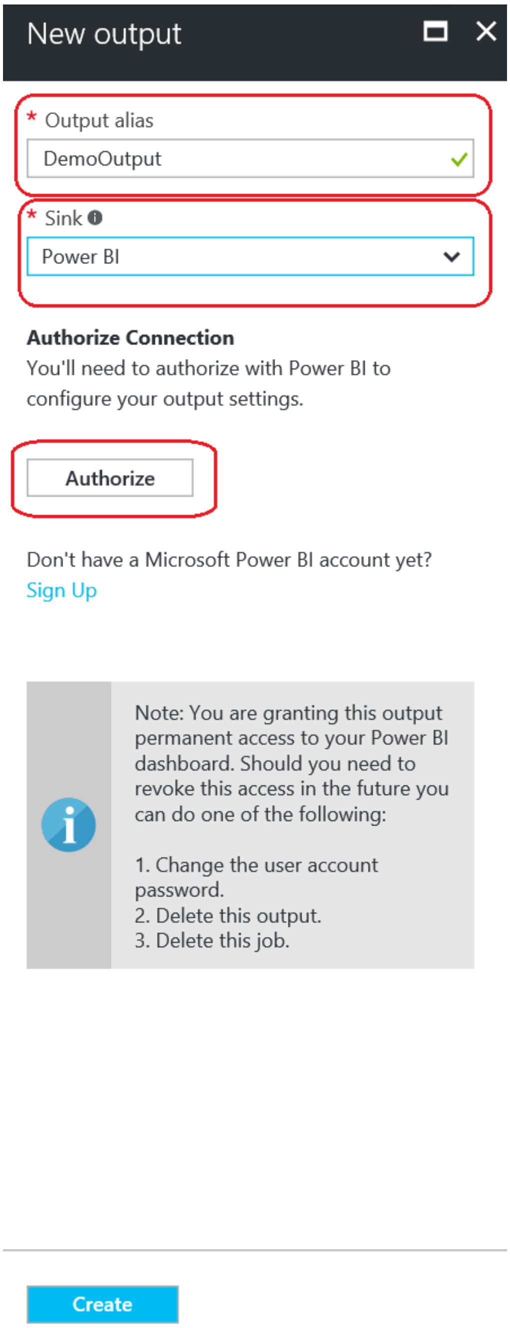
Settings for New Output Blade

Output alias
PowerBI

Sink
Select the Power BI

Authorize
Your Power BI credential

Click Authorize and you will be redirected to the Microsoft login page to authorize output to your Power BI account.



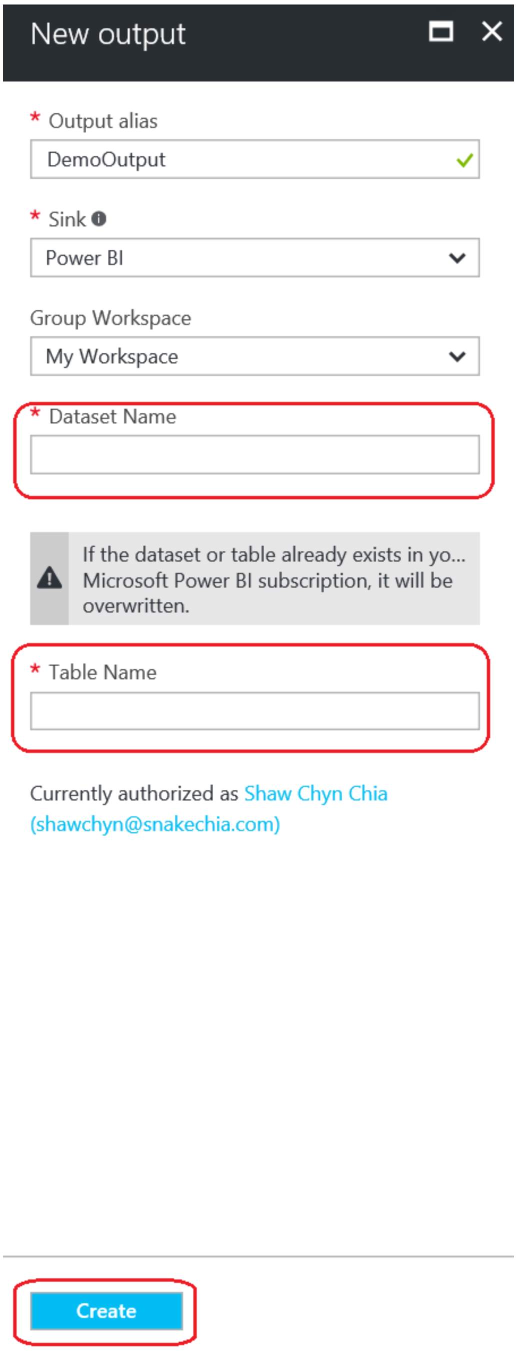
2. Once Power BI is authorized, continue setting the remaining configuration fields.

<u>Dataset Name</u>
PowerBI

<u>Table Name</u>
SenseHatTelemetry

<u>Click Create</u>
It will take a moment or two to create the Output stream.

Close the Output Blade



Stream Analytics – Query Configuration

Now the job's inputs, outputs are configured, the Stream Analytics Job needs to know how to transform the input data into the output data source. To do so, you will create a new Query.

1. From Stream Analytics Configuration blade, click **Query**.

2. Replace the default query with the following Stream Analytics Query.
   Note: We just take everything send from the devices and dump it into the output.

   Need help with your query? Check out some of the most common Stream Analytics query patterns here.

```
SELECT *
INTO [DemoPBiOutput]
FROM [DemoInput]
```

3. Click the **SAVE** button and confirm.

4. **Close** the **Query** blade.

Starting the Stream Analytics Job

The job is configured and it now needs to be started.

1. From the Stream Analytics configuration blade, click
   Start -> Now -> Start.
   Allow for 30 – 60 seconds for the job to enter "**Running**" mode.

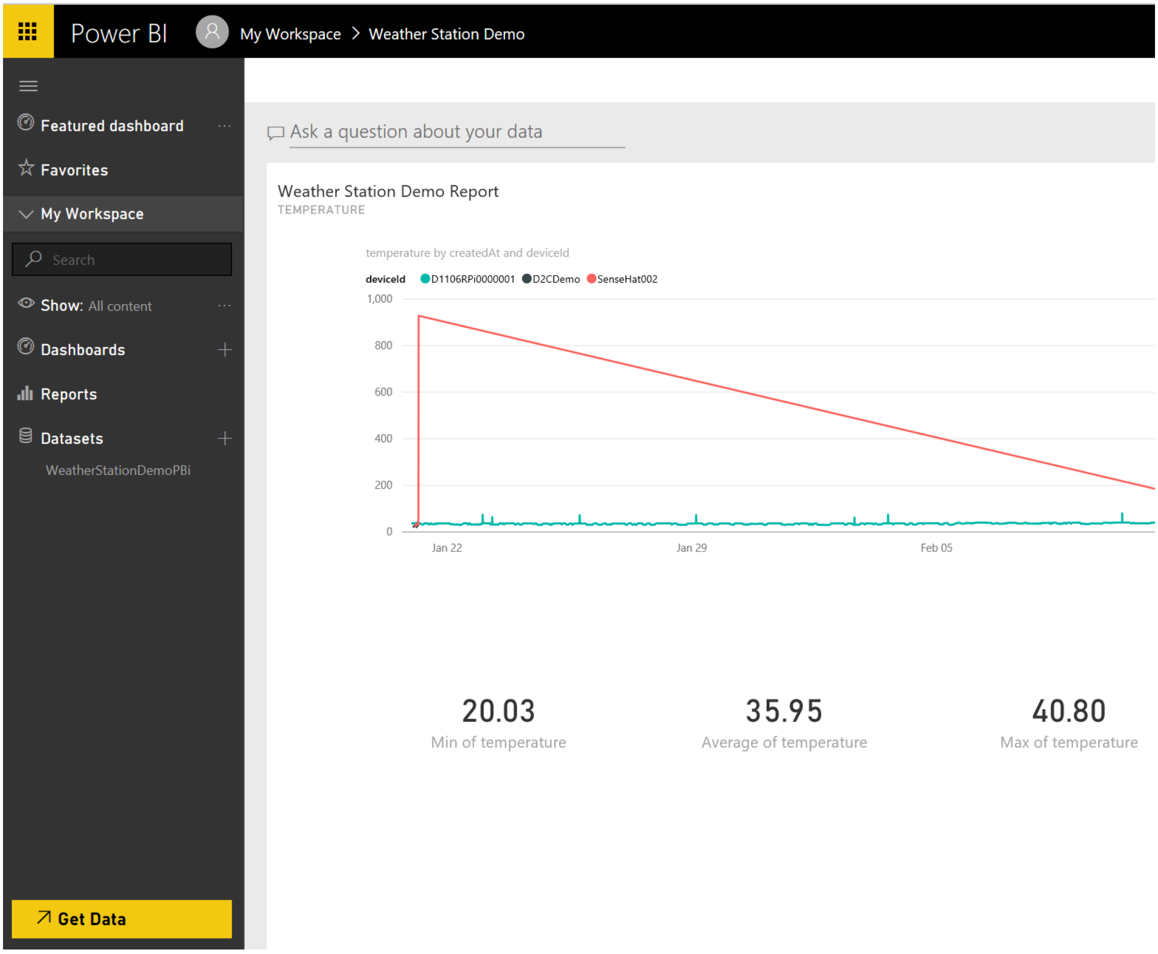
   Once the job starts and it is receiving data from your IoT devices, it will create the Power BI datasource associated with the given subscription.

Lab 10: Microsoft Power BI

Power BI transforms your data into rich visuals for you to collect and organize so you can focus on what matters to you.

Setting Up the Power BI Dashboard

1. Navigate to **Power Bi** (http://www.powerbi.com) and authenticate. Click the **Hamburger** to expand the navigation pane.
   The **Stream Analytics job** needs to run for few minutes before it appears in the navigation pane under the **Datasets**.

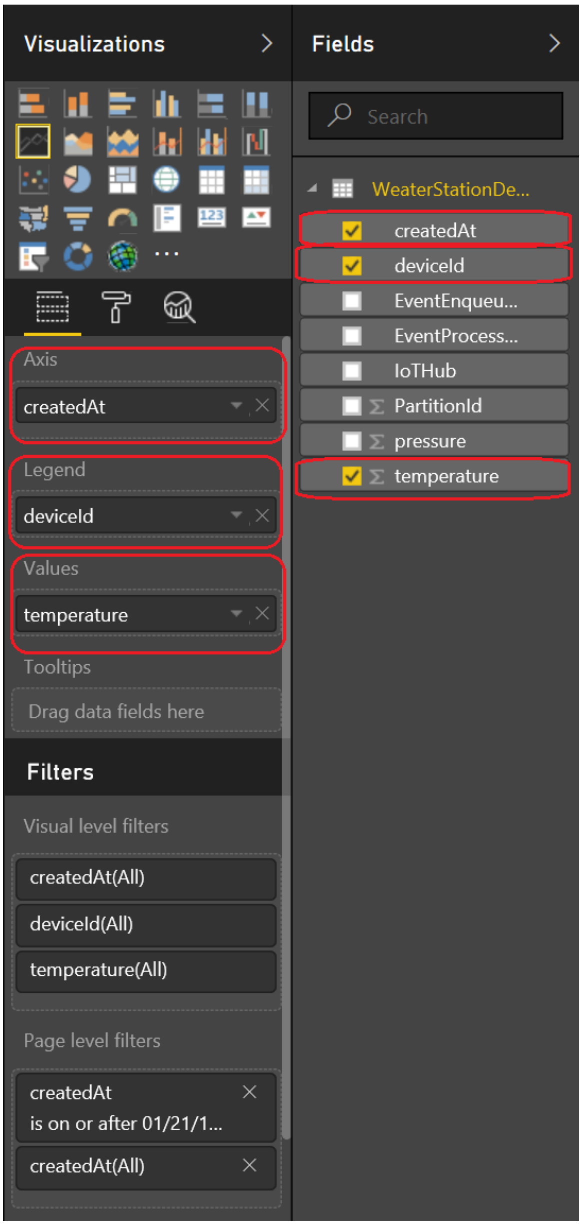


The Power BI dataset will ONLY be created if the job is running and if it is receiving data from the IoT Hub input. If there is no dataset then check if the Weather Station App is running on your Raspberry Pi device and it is streaming data to Azure IoT Hub. To verify the Stream Analytics job is receiving and processing data you can check the Azure Management Stream Analytics monitor.

Defining A Power BI Report

Click on the datasource name that you created and start defining the report.

The Report designer will be opened showing the list of fields available for the selected datasource and the different visualizations supported by the tool.

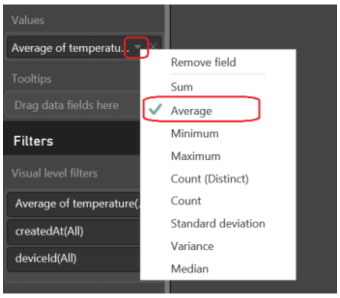


1. Select **Line Chart** from the **Visualizations** and ensure it is selected in the designer.

2. Drag and drop the following fields from the Fields section:

    - deviceId -> Legend
    - createdAt -> Axis
    - temperature -> Values

3. Select Sum from the Values dropdown menu.



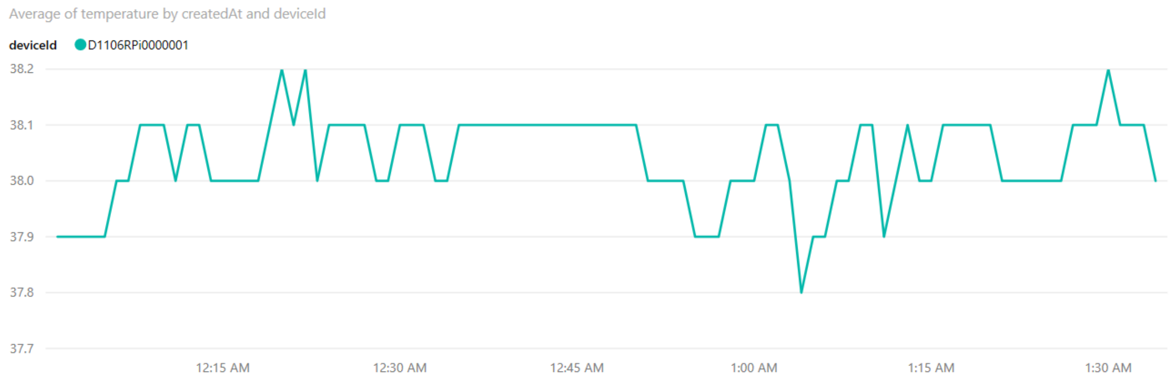
4. From the deviceId(All) dropdown select your device.

5. Click the **SAVE** button and set *Temperature By Time* as the name of the Report.

6. Repeat Step 1 to 5 for Pressure and Humidity.

7. Now create a new Dashboard, and pin the report to it. Click the plus sign (**+**) next to the **Dashboard** section to create a new dashboard and name it **Weather Station**.

8. Go back to your report and click the Pin Live Page icon to add the reports to the newly created dashboard.



Congratulations!

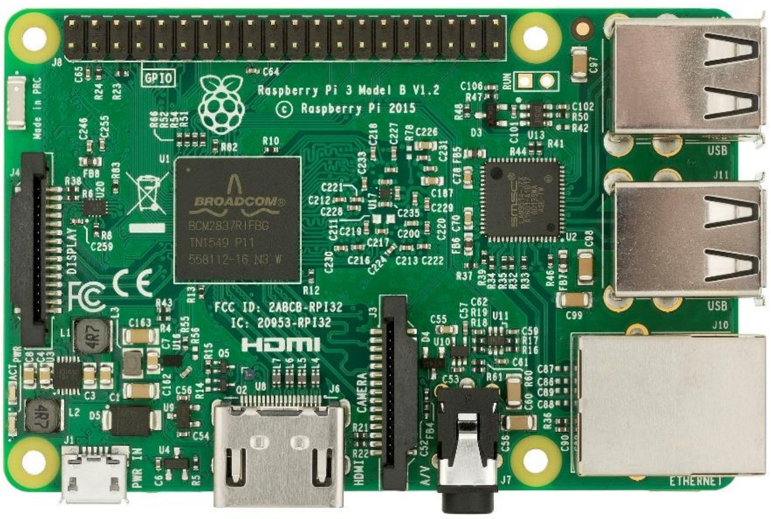
You have finished all the lab exercises.

- You have install and configure Windows 10 IoT Core for Raspberry Pi 2/3 device.
- You have successfully deploy Universal Windows Apps to Raspberry Pi 2/3.
- You have successfully read/write data thru GPIO pins on Raspberry Pi 2/3.
- You have successfully create a mini Weather Station with Sense HAT for Raspberry Pi.
- You have successfully stream the weather data to Azure IoT Hub.
- You have successfully received message from Azure IoT Hub to Raspberry Pi 2/3.
- You have successfully stream data from Azure IoT Hub to Power BI using Azure Stream Analytics.
- You have successfully visualized data with Power BI.

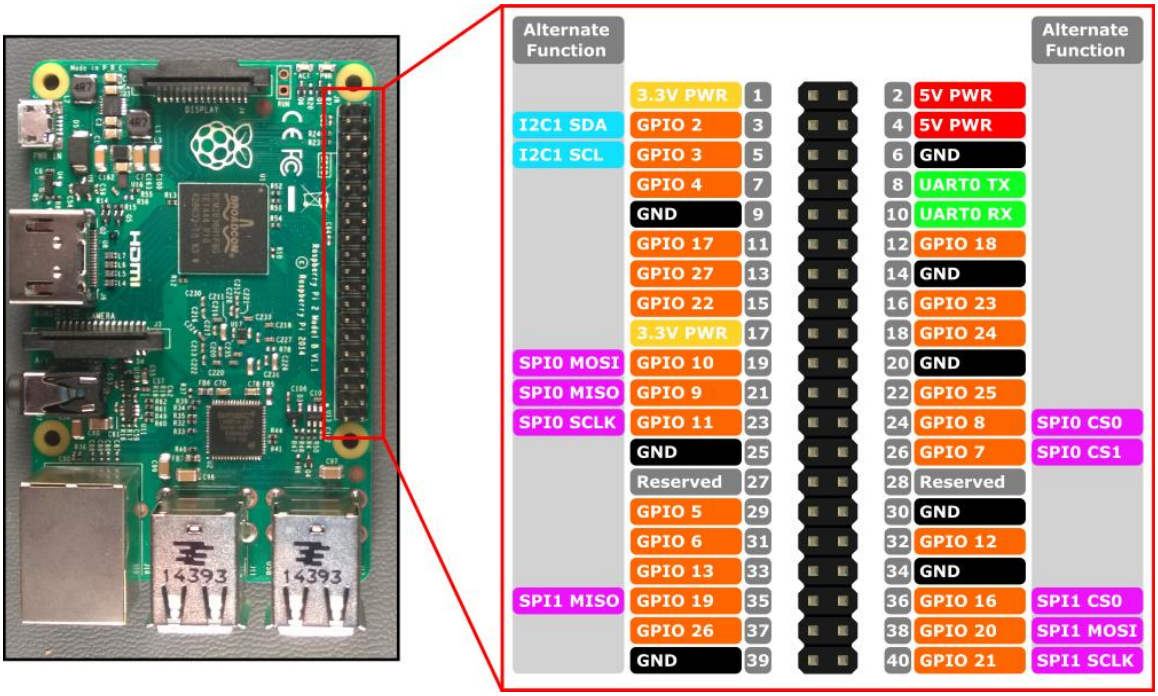
Appendix

Raspberry Pi 2/3 Specification

<table>
<tr><th></th><th>Model B Gen 2</th><th>Model B Gen 3</th></tr>
<tr><td>Release date</td><td>February 2015</td><td>February 2016</td></tr>
<tr><td>Architecture</td><td>ARM v7-A (32-bit)</td><td>ARM v80A (64-bit/32-bit)</td></tr>
<tr><td>SoC</td><td>Broadcom BCM2836</td><td>Broadcom BCM2837</td></tr>
<tr><td>CPU</td><td>900 MHz 32-bit quad-core ARM Cortex-A7</td><td>1.2 GHz 64-bit quad-core ARM Cortex-A53</td></tr>
<tr><td>GPU</td><td colspan="2">Broadcom VideoCore IV @ 250 MHz</td></tr>
<tr><td>Memory</td><td colspan="2">1 GB (shared with GPU)</td></tr>
<tr><td>USB 2.0 ports</td><td colspan="2">4 (via the on-board 5-port USB hub</td></tr>
<tr><td>Video Input</td><td colspan="2">15-pin MIPI camera interface (CSI) connector, used with the Raspberry Pi camera or Raspberry Pi NoIR camera</td></tr>
<tr><td>Video Output</td><td colspan="2">HDMI (rev 1.3), composite video (3.5 mm TRRS jack), MIPI display interface (DSI) for raw LCD panels</td></tr>
<tr><td>Audio Input</td><td colspan="2">via I^2S</td></tr>
<tr><td>Audio Output</td><td colspan="2">Analog via 3.5 mm phone jack; digital via HDMI</td></tr>
<tr><td>On-board Storage</td><td>MicroSDHC slot</td><td>MicroSDHC slot, USB Boot Mode</td></tr>
<tr><td>Onboard Network</td><td>10/100 Mbit/s Ethernet (8P8C) USB adapter on the USB hub</td><td>10/100 Mbit/s Ethernet, 802.11n wireless, Bluetooth 4.1</td></tr>
<tr><td>Low Level Peripherals</td><td colspan="2">17× GPIO plus the same specific functions, and HAT ID bus</td></tr>
<tr><td>Power Ratings</td><td colspan="2">800 mA (4.0 W)</td></tr>
<tr><td>Power Source</td><td colspan="2">5 V via MicroUSB or GPIO header</td></tr>
<tr><td>Size</td><td colspan="2">85.60 mm × 56.5 mm (3.370 in × 2.224 in)</td></tr>
<tr><td>Weight</td><td colspan="2">45 g (1.6 oz)</td></tr>
<tr><td>Console</td><td colspan="2">Micro-USB cable or a serial cable with optional GPIO power connector</td></tr>
</table>



Raspberry Pi 2/3 GPIO Layout



SenseHat Fact Sheet

The Raspberry Pi Sense HAT is attached on top of the Raspberry Pi via the 40 GPIO pins (which provide the data and power interface) to create an 'Astro Pi'. The Sense HAT has several integrated circuit based sensors that you can use for many different types of experiments, applications, and even games.

Technical Specification

- Gyroscope – angular rate sensor: ±245/500/2000dps
- Accelerometer - Linear acceleration sensor: ±2/4/8/16 g
- Magnetometer - Magnetic Sensor: ±4/8/12/16 gauss
- Barometer: 260 – 1260 hPa absolute range (accuracy depends on the temperature and pressure, ±0.1 hPa under normal conditions)
- Temperature sensor (Temperature accurate to ± 2 °C in the 0-65 °C range)
- Relative Humidity sensor (accurate to ±4.5% in the 20-80%rH range, accurate to ± 0.5 °C in 1540 °C range)
- 8x8 LED matrix display
- Small 5 button joystick

Others

Useful Network Commands

From PowerShell

- netsh wlan show profile
- netsh wlan add profile *Wi-Fi-ProfileName.xml*
- netsh wlan export profile key=clear
- netsh wlan delete profile *ProfileName*
- netsh wlan connect name=*ProfileName*
- netsh wlan show interfaces
- netsh interface ipv4 set dns “Wi-Fi” static <IP address>
- netsh interface ipv4 set address “Wi-Fi” static <IP address> <space> <subnet> <space> <gateway address>

Source Code

https://github.com/snakechia/Windows10IoTAzureIoTLabs/

Social

#windows10 #windows10iot #windows10iotcore #azure #azureiothub #iot #raspberrypi

Disclaimer

All care has been taken to ensure the accuracy of this document. No liability accepted.

Copyright