

## INTERNET OF THINGS MAKER DEN LAB GUIDE



Windows 10 IoT Core

Internet of Things

Maker Den FEZ HAT Lab Guide

Document Version 3.0

Social	<b>Twitter</b> #makerden #iot #raspberrypi #windows10 #azure
Document Authors	Dave Glover   <a href="mailto:dglover@microsoft.com">dglover@microsoft.com</a>   @dglover Andrew Coates   <a href="mailto:acoat@microsoft.com">acoat@microsoft.com</a>   @coatsy Fai Lai   <a href="mailto:hoongfai@microsoft.com">hoongfai@microsoft.com</a>   @faister
Document location	<a href="https://github.com/MakerDen/Maker-Den-Documentation-and-Resources-FezHat">https://github.com/MakerDen/Maker-Den-Documentation-and-Resources-FezHat</a>
Source Code Location	<a href="https://github.com/MakerDen/Maker-Den-Windows-IoT-Core-FEZ-HAT">https://github.com/MakerDen/Maker-Den-Windows-IoT-Core-FEZ-HAT</a>
Disclaimer	All care has been taken to ensure the accuracy of this document. No liability accepted.
Copyright	You are free to reuse and modify this document and associated software.

## INTRODUCTION

The goal of the Maker Den is to familiarise you with some of the components and technologies associated with the Internet of Things (IoT). Along the way, you will experience deploying code, and streaming sensor data to Microsoft Azure, aggregating data with Stream Analytics and reporting with Microsoft Power Bi.

## GETTING STARTED

If you are setting up your own Maker Den then all source code and documentation is available at <https://github.com/MakerDen/Maker-Den-Documentation-and-Resources-FezHat>.

## TIME REQUIRED

There are two sections to this lab. The first section is device centric and will take less than 15 minutes. Section 2 is more cloud centric and will take approximately an hour. You are more than welcome to stay longer and delve a little deeper.

## SPREAD THE WORD

Be sure to spread the word about the Internet of Things Maker Den on Twitter. Use hash tags #makerden #iot #raspberrypi #windows10 #azure

## LAB HARDWARE

The following components are used for the Maker Den.

### Raspberry Pi 2

These labs are built on the Raspberry Pi running Windows 10 IoT Core.

You can find out more about Windows 10 IoT Core at <http://dev.windows.com/iot>.



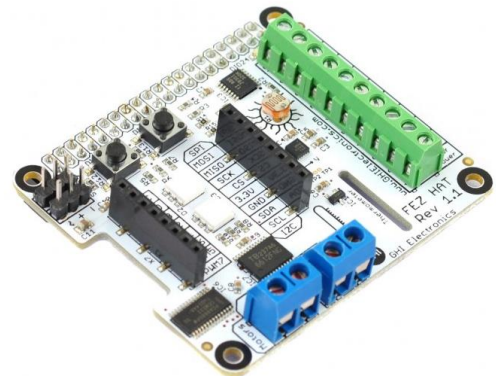
### GHI electronics FEZ HAT

The FEZ HAT Key Features:

- On-Board Analog Input and PWM chips.
- Two DC Motor Drivers, suitable for building small robots.
- Terminal Blocks for wiring in DC motors without the need for soldering.
- Two Servo Motor Connections.
- Two Multi Color LEDs, connected to PWM for thousands of colors.
- Single Red LED.
- Temperature Sensor.
- Accelerometer.
- Light Sensor.
- Two user buttons.
- Terminal block with 2x Analog, 2x Digital I/O, 2x PWM and power.
- Female headers with SPI, I2C, 3x Analog, 3x PWM.
- Dedicated power input for driving the servo motors and DC motors.
- No Soldering required, completely assembled and tested.

#### Developer Guide

<https://www.ghielectronics.com/docs/329/fez-hat-developers-guide>



---

## RESET THE LAB

✕ STEP 1: Ensure Visual Studio is closed.

✕ STEP 2: Double click the **ResetLabs.bat** file on your desktop. This will copy the source code from a GitHub repository and launch Visual Studio with the solution opened.

## EXPERIMENTS

✕ There are ten Maker Den experiments to get you started with Windows 10 IoT Core and Microsoft Azure IoT Services.

✕ All the source code can be referenced from the Source Code folder on the Desktop.

✕ This user guide and an architectural overview of the Maker Den can be found in the Documents folder on the Desktop.

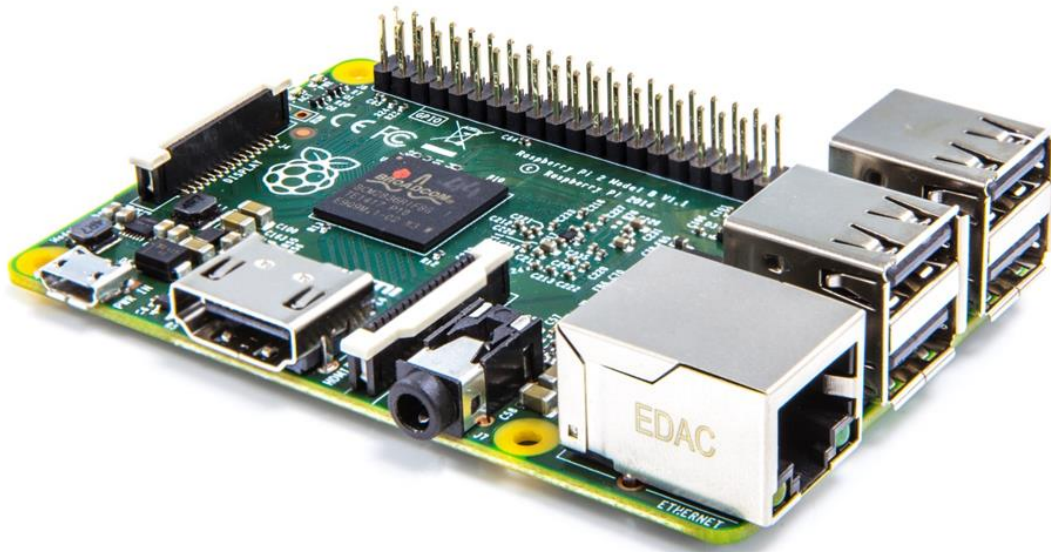
✕ Be sure to check out the [Windows 10 IoT Core Doc, Tutorials and Samples](#). There is a link to this page in the Desktop Documents folder.

✕ For the self-sufficient adventurous types, you can reference the [Windows 10 IoT Core Doc, Tutorials and Samples](#) and the [GHI Electronics FEZ HAT](#) developer resources for more information.

---

# Section 1

---



## **Windows IoT Core development with Visual Studio**

Connecting up your device

Deploying your first app

Sensing ambient light levels

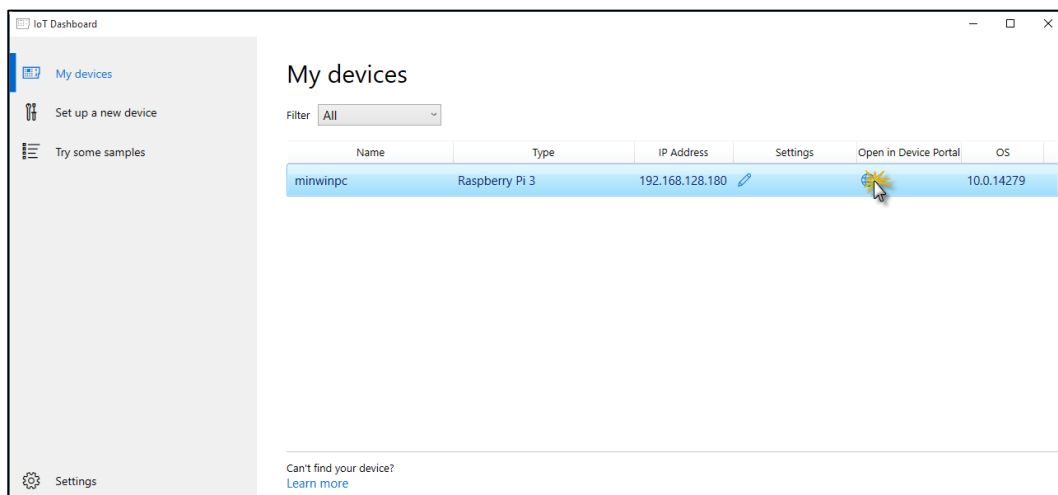
---

## EXPERIMENT 1: CONNECTING AND CONFIGURING YOUR DEVICE

The Raspberry Pi should be connected to the development PC via a wired Ethernet connection. This connection is used both for deployment and debugging as well as passing through internet requests from the Raspberry Pi when [Internet Connection Sharing](#) is enabled on the PC.

☒ **STEP 1:** Press the Windows key and type “Windows 10 IoT Core Dashboard”<sup>1</sup> and run.

☒ **STEP 2:** Go to My devices<sup>2</sup> and click the Open in Device Portal icon of your device name.



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 public and Device Discovery is not enabled. See [How to change Windows 10 network location from Public to Private](#).

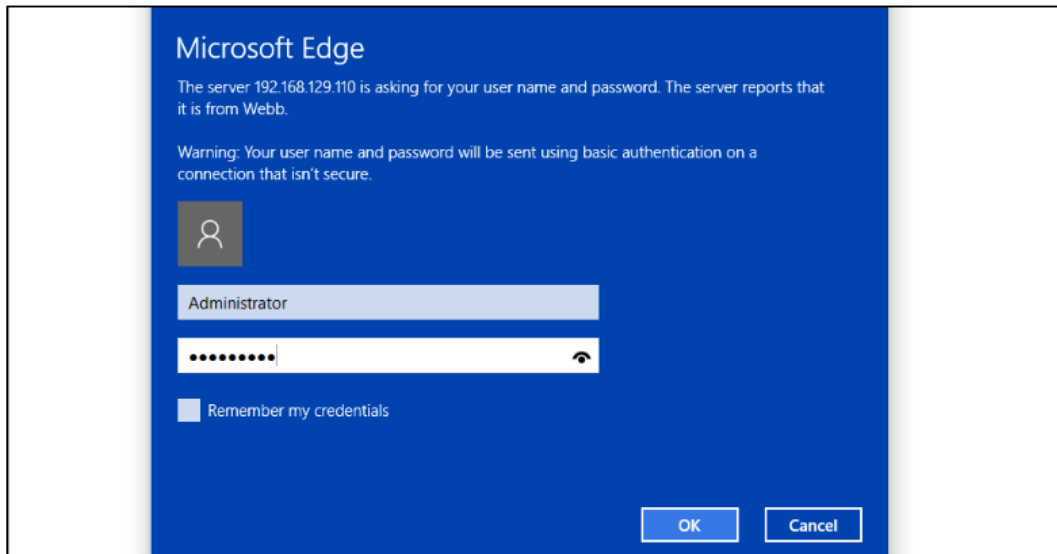
Alternatively, navigate to the default device url <http://minwinpc:8080>.

---

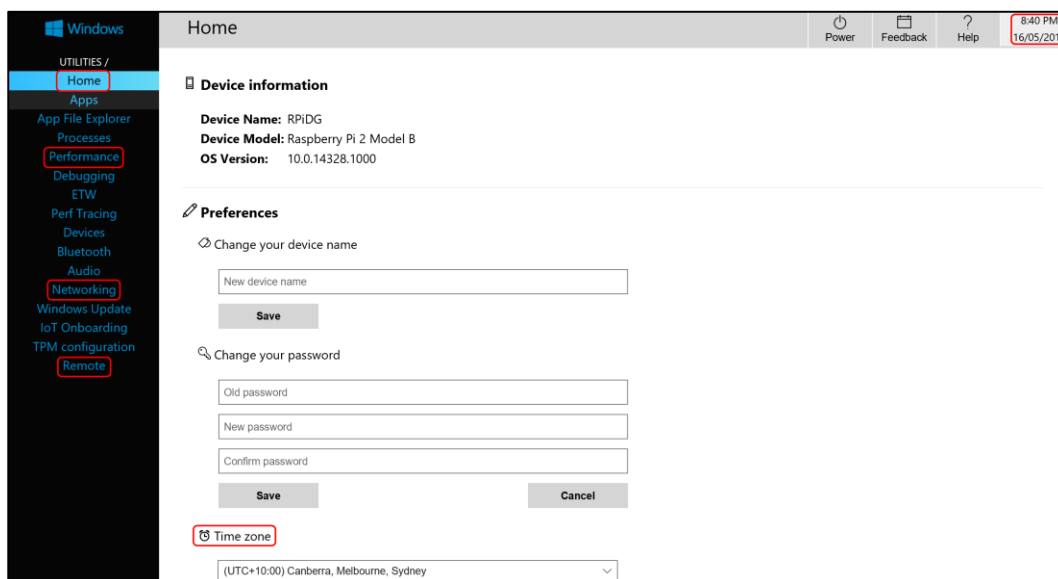
<sup>1</sup> You can download the Windows 10 IoT Core Dashboard from <https://developer.microsoft.com/en-us/windows/iot/getstarted>

<sup>2</sup> You can right mouse click a device for more options including copying the device IP Address, Name, plus start a PowerShell session.

✕ **STEP 3:** Authenticate. The default credentials are Username: *Administrator* and Password: *p@ssw0rd*



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



✕ **STEP 4:** Verify Device Configuration

- From the **Home** Tab verify the Time Zone, date and time are correct. If the device has the incorrect data or time, then refer to the [troubleshooting](#) section in the appendix.

- From the **Remote** tab verify that **Windows IoT Remote Server**<sup>3</sup> enabled. If it is not, then enable it.
- Take a moment to explore the other tabs in the Windows Device Portal.

✉ **STEP 5:** Test Windows IoT Remote Client connection.

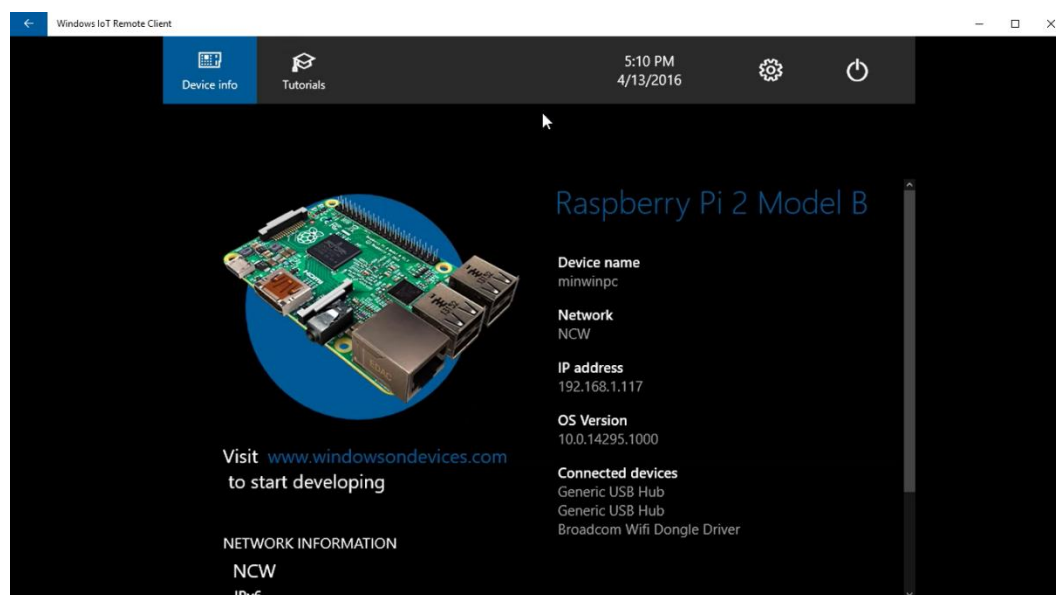
Press the Windows key and type “Windows IoT Core Remote Client”<sup>4</sup> and run.

✉ **STEP 6:** Select your device from the dropdown list.

Depending on the network setup you may need to enter the IP address of your Raspberry Pi. Get the address of the device from the **Windows 10 IoT Core Dashboard**.

This will take a moment to connect. When it does you will see the video output of the Raspberry Pi remoted to your desktop.

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



<sup>3</sup> The Windows IoT Remote Server does take additional CPU cycles on the Raspberry Pi so depending on what you are doing you may want to disable the Windows Remote Server from the Windows Device Portal.

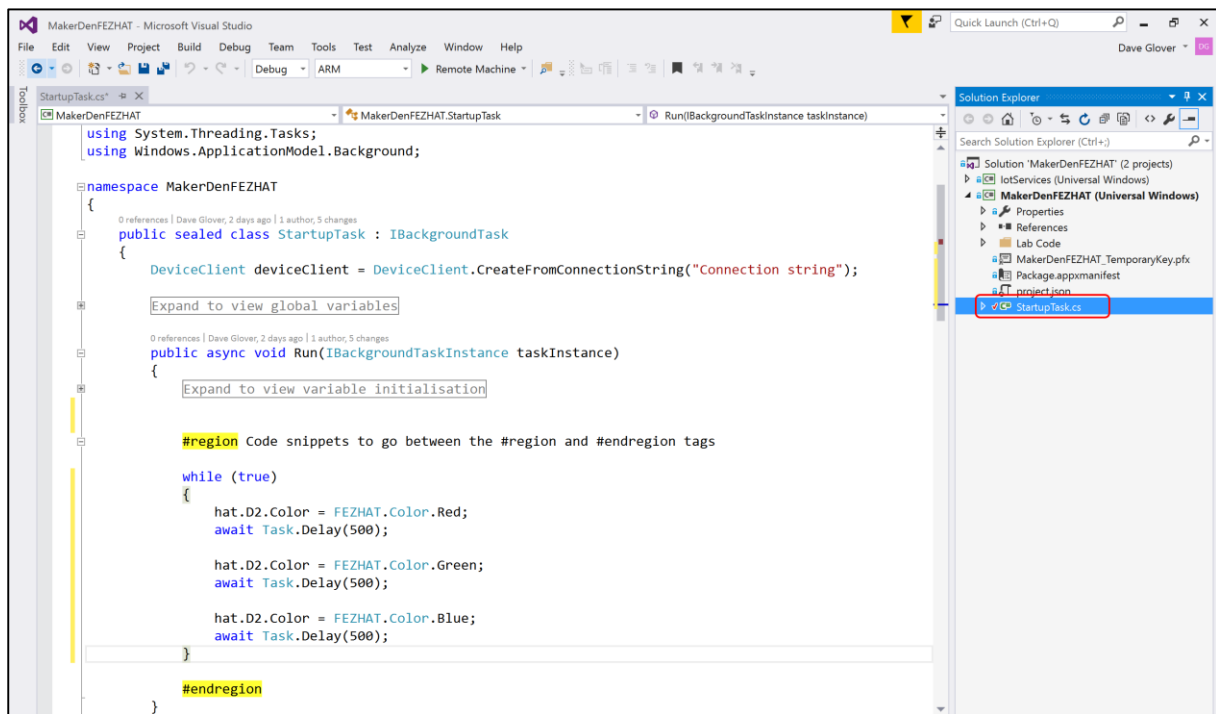
<sup>4</sup> The [Windows IoT remote Client](#) is available from the Windows Store.



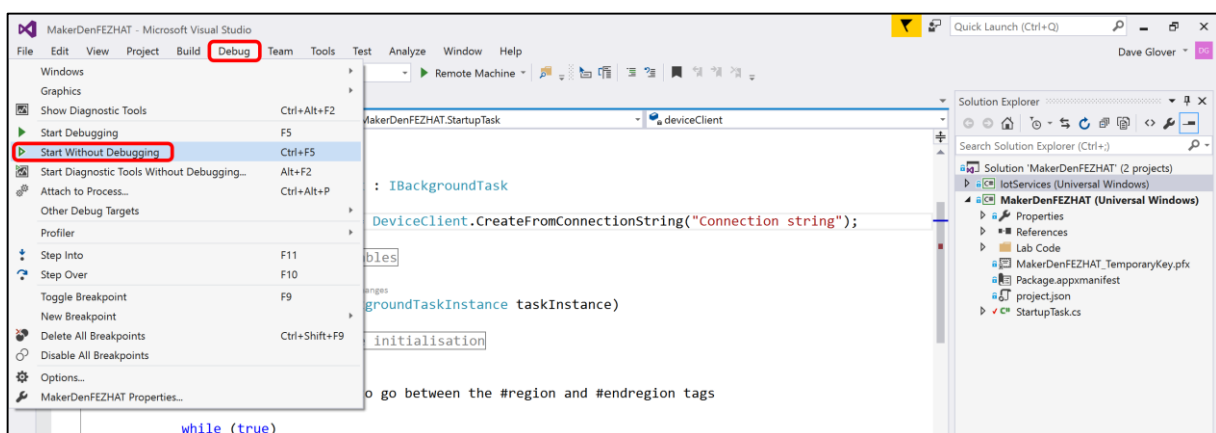
## EXPERIMENT 2: HELLO WORLD

Deploy your first experiment to ensure everything is setup correctly and to check Visual Studio is communicating with your Raspberry Pi.

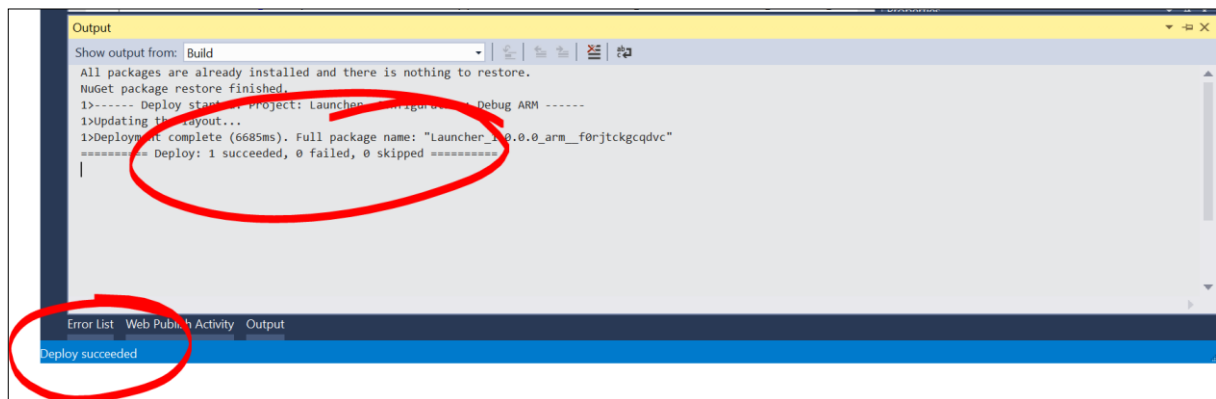
☒ **STEP 1:** Expand the **MakerDen** project then double click the **StartupTask.cs** file to open it.



☒ **STEP 2:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start Without Debugging** or from the keyboard press **Ctrl+F5**.



☒ **STEP 3:** Check that Visual Studio has successfully compiled and deployed the code by looking at the output window and the status bar.



☒ **STEP 4:** Check the LEDs on the FEZ HAT. You should see an LED alternating between Red, Green and Blue.

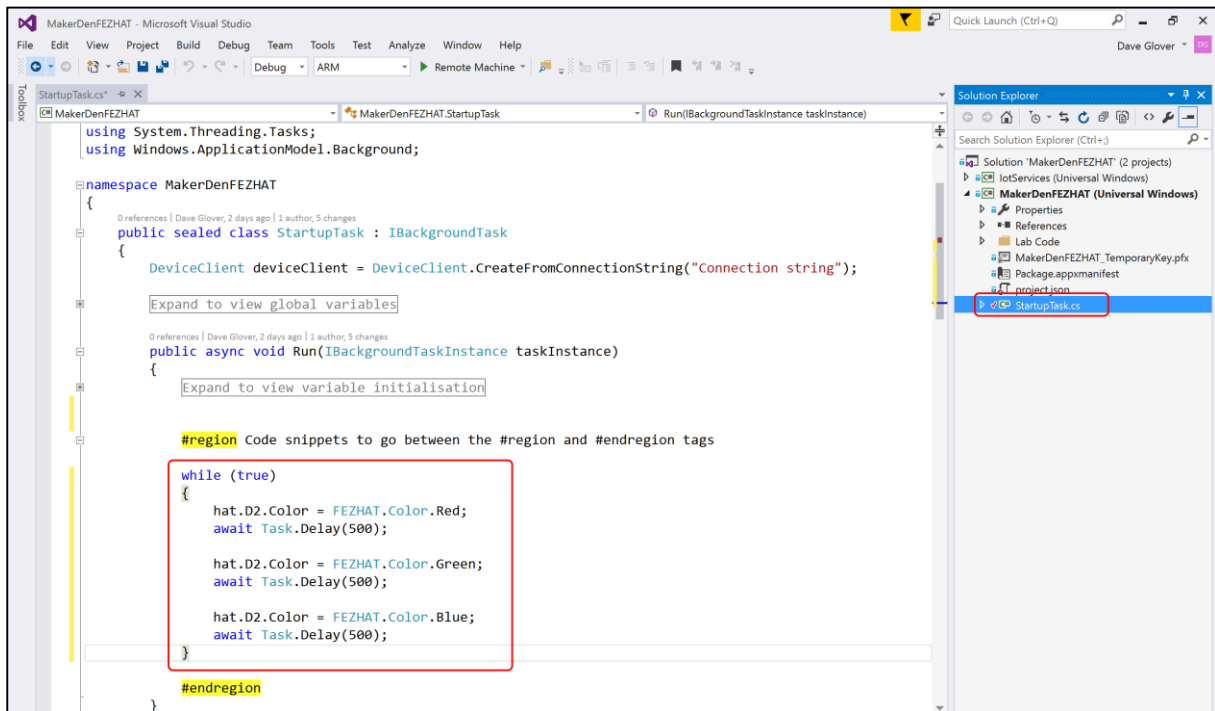
☒ **STEP 5:** Pat yourself on the back, you did it 😊

## EXPERIMENT 3: SENSING THE WORLD

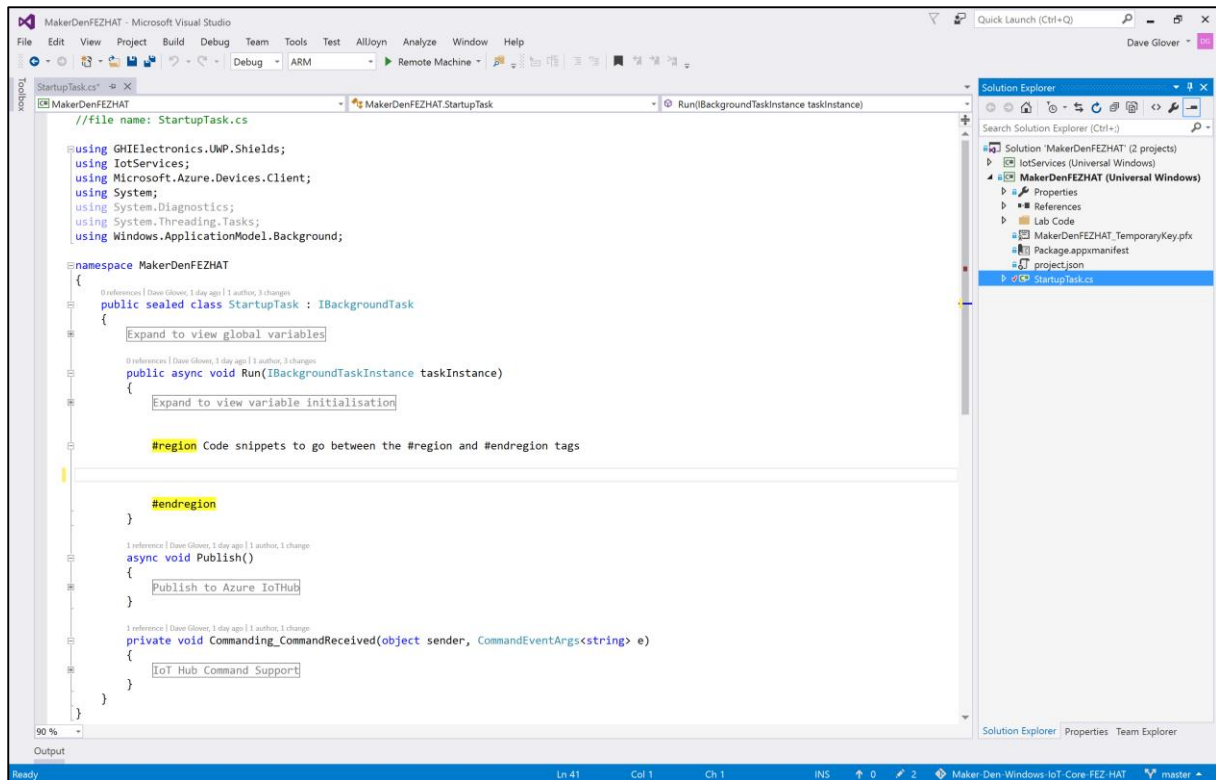
This lab reads the ambient light levels from the light sensor.

✕ **STEP 1:** Review the code in the **StartupTask.cs** file. Look for the **#region** and **#endregion** tags.

✕ **STEP 2:** Delete the code circled in red **inside** the **#region** tags.



Your “StartupTask.cs” file should look like the screenshot below after you have deleted the code. If it doesn’t look the same then **Ctrl+Z** to undo the changes you made and try again.



✘ **STEP 3:** Type the following code between the #region tags **OR** using a code snippet type **lab3** and press Tab twice.

```
while (true)
{
    var level = hat.GetLightLevel() * 100;

    if (hat.GetLightLevel() * 100 > LIGHT_THRESHOLD)
    {
        hat.D2.Color = FEZHAT.Color.Blue;
    }
    else
    {
        hat.D2.Color = FEZHAT.Color.Red;
    }

    await Task.Delay(500);
}
```

✘ **STEP 4:** Your “StartupTask.cs” file should look like the following. If not, **Ctrl+Z** and try again.

```
//file name: StartupTask.cs

using GHIElectronics.UWP.Shields;
using IotServices;
using Microsoft.Azure.Devices.Client;
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.ApplicationModel.Background;

namespace MakerDenFEZHAT
{
    public sealed class StartupTask : IBackgroundTask
    {
        DeviceClient deviceClient = DeviceClient.CreateFromConnectionString("Connection String");

        Expand to view global variables

        public async void Run(IBackgroundTaskInstance taskInstance)
        {
            Expand to view variable initialisation

            #region Code snippets to go between the #region and #endregion tags

            while (true)
            {
                var level = hat.GetLightLevel() * 100;

                if (hat.GetLightLevel() * 100 > LIGHT_THRESHOLD)
                {
                    hat.D2.Color = FEZHAT.Color.Blue;
                }
                else
                {
                    hat.D2.Color = FEZHAT.Color.Red;
                }

                await Task.Delay(500);
            }

            #endregion
        }

        async void Publish()
        {
            #region Publish to Azure IoT Hub
            #endregion
        }

        private void Commanding_CommandReceived(object sender, CommandEventArgs<string> e)
        {
            #region IoT Hub Command Support
            #endregion
        }
    }
}
```

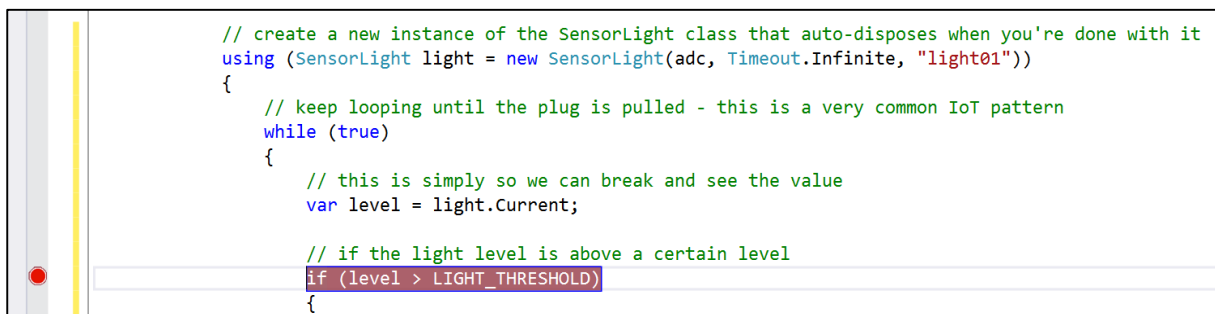
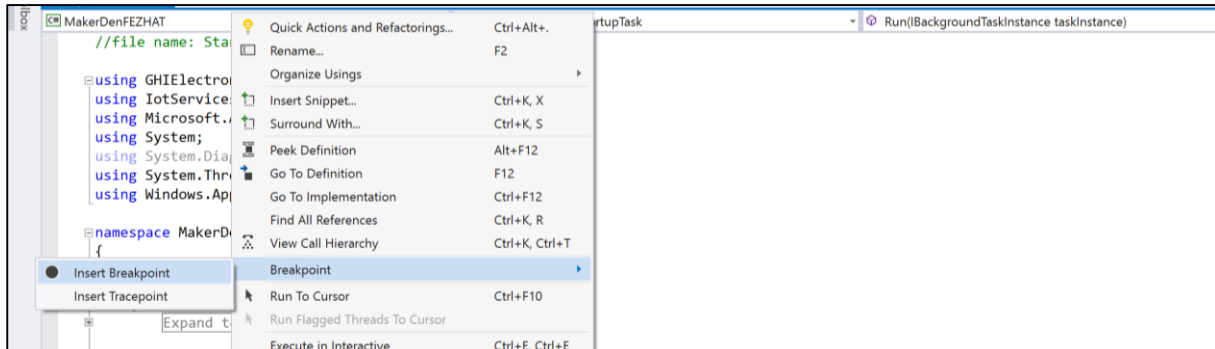
☒ **STEP 5:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start Without Debugging** or from the keyboard press **Ctrl+F5**.

☒ **STEP 6:** Hover your hand over the light sensor and observe the LED alternates between blue and red depending on the ambient light levels.

## EXPERIMENT 4: REMOTE DEBUGGING

☒ **STEP 1:** Next, set a break point to see how easy it is to debug directly on the device. This is a unique capability provided by Visual Studio and Windows IoT Core.

- Right-click on the line that reads `if (level > LIGHT_THRESHOLD)`
- Choose Breakpoint, then Insert Breakpoint.



☒ **STEP 2:** From the **Debug** menu select **Start Debugging** or on the keyboard press **F5** and wait for the solution to deploy and for Visual Studio to hit the breakpoint.

☒ **STEP 3:** Hover the cursor over the variable “level” and Visual Studio will display its current value.

☒ **STEP 4:** While holding your hand over the light sensor, press F5 a couple of times to continue and observe the LED changes colour depending on ambient light levels.

☒ **Step 5:** Press Shift-F5 to stop debugging.

---

# Section 2

---



## **Microsoft Azure Cloud Development**

Provisioning IoT Hub and a device

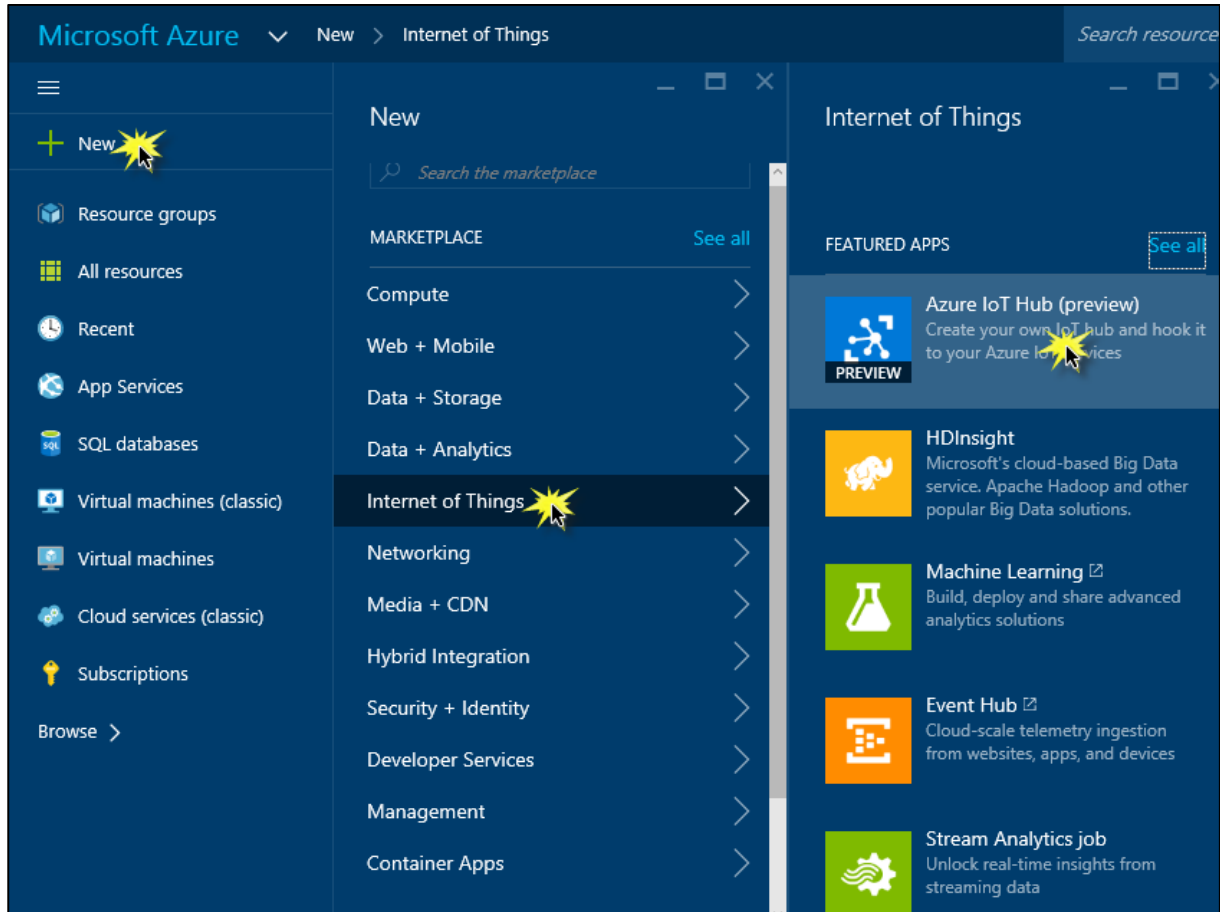
Stream Analytics

Power BI

## EXPERIMENT 5: PROVISIONING AN AZURE IOT HUB AND AN IOT DEVICE

✳️ **STEP 1:** Sign in to your Azure portal<sup>5</sup> at <http://portal.azure.com>

✳️ **STEP 2:** Create a new IoT Hub. Click **New** in the Jumpbar, then click **Internet of Things**, then click **Azure IoT Hub**.



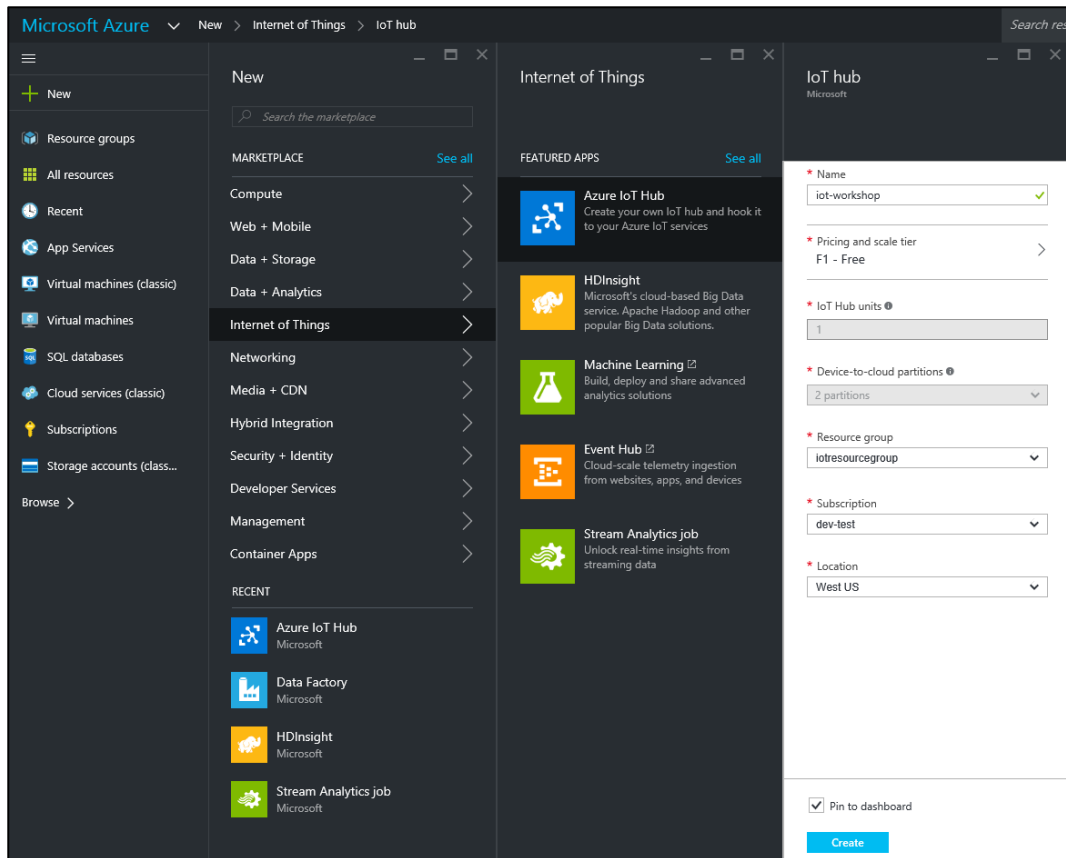
✳️ **STEP 3:** Configure the **IoT Hub** as follows :-

- Enter a **Name** for the hub e.g. iot-workshop (must be global unique name)
- Select a **Pricing and scale tier** (F1 FREE tier is enough for the lab)
- Create a new resource group, or select an existing one. For more information, see [Using resource groups to manage your Azure resources](#).
- Select the **Region** closest to where your solution and or devices will be located.

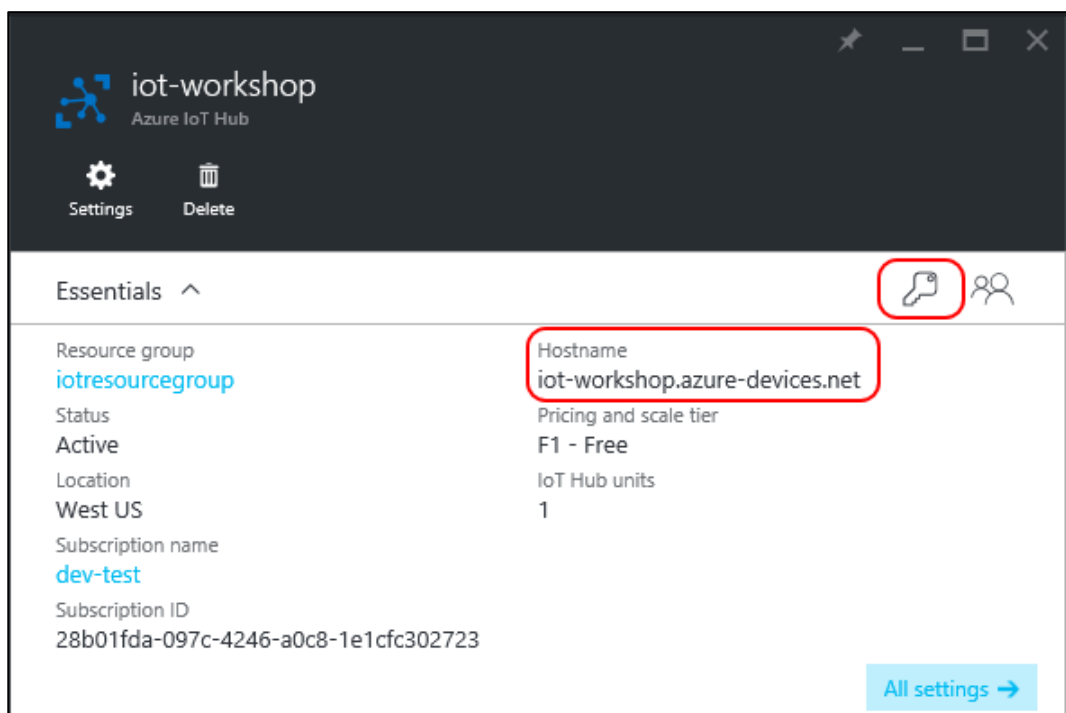
---

<sup>5</sup> See options for [Provisioning an Azure Account](#).





✳️ **STEP 4:** It can take a few minutes for the IoT hub to be created. Once it is ready, open the blade of the new IoT hub, take note of the URI and select the key icon at the top to access to the shared access policy settings.



✳️ **STEP 6:** Select the **iothubowner** Shared access policy. Click the **Connection string—primary key** copy icon to copy the connection string to the clipboard.

POLICY	PERMISSIONS
iothubowner	registry write, service connect, device connect
service	service connect
device	device connect
registryRead	registry read
registryReadWrite	registry write

**iothubowner**  
iot-workshop

Save Discard Regen key Delete

Access policy name  
iothubowner

Permissions

- ☒ Registry read ⓘ
- ☒ Registry write ⓘ
- ☒ Service connect ⓘ
- ☒ Device connect ⓘ

Shared access keys

Primary key ⓘ  
6ZUCGTkhXqolrSCbrATiIDUC8jvm91nl6B...

Secondary key ⓘ  
R5Ppllohcm5pJL26Lybx3PaMpGYk/HYm5

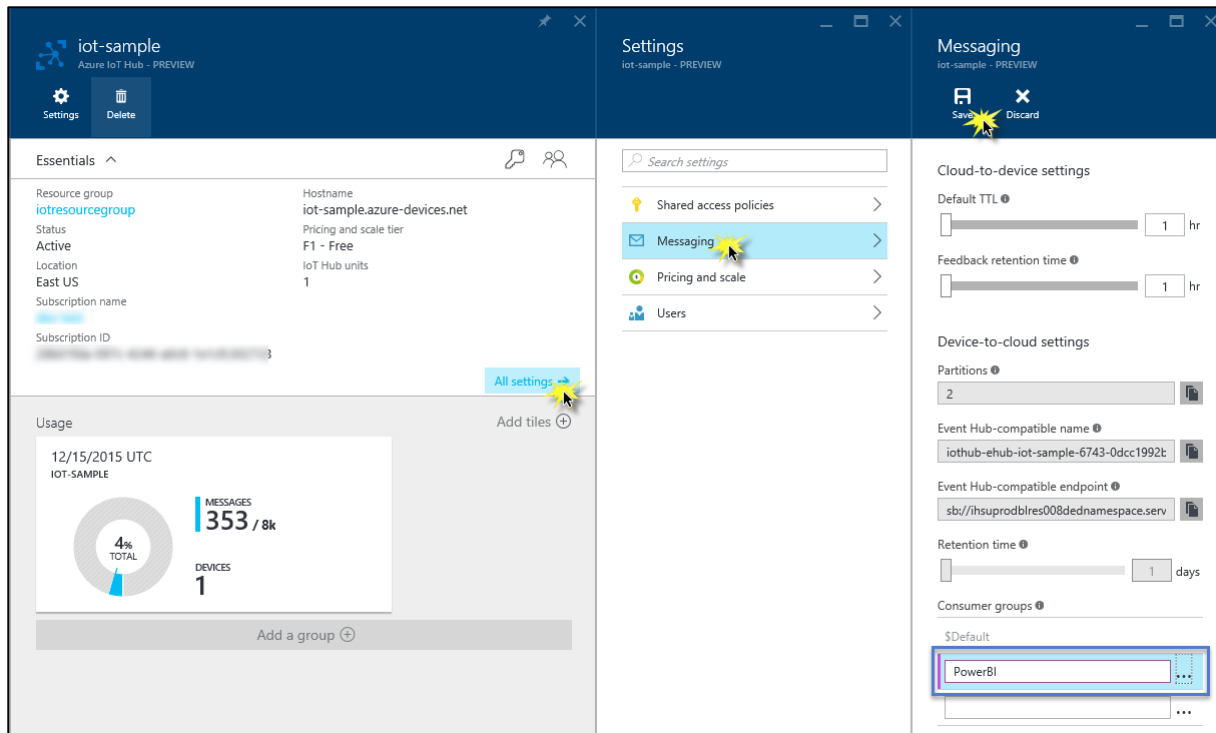
Connection string—primary key ⓘ  
HostName=iot-workshop.azure-devices.r...

Connection string—secondary key ⓘ  
HostName=iot-workshop.azure-devices.r...

## CREATE A SERVICE BUS CONSUMER GROUP

For Experiment 7 we need an additional consumer group to allow several applications to independently read data from the IoT Hub. Follow these steps :-

✳️ **STEP 1:** From the IoT Hub blade, click **All settings** and then on **Messaging**.



✖ **STEP 2:** At the bottom of the Messaging blade, type the name of the new Consumer Group "powerbi".

✖ **STEP 3:** From the top of the Messaging menu, click on the Save icon.

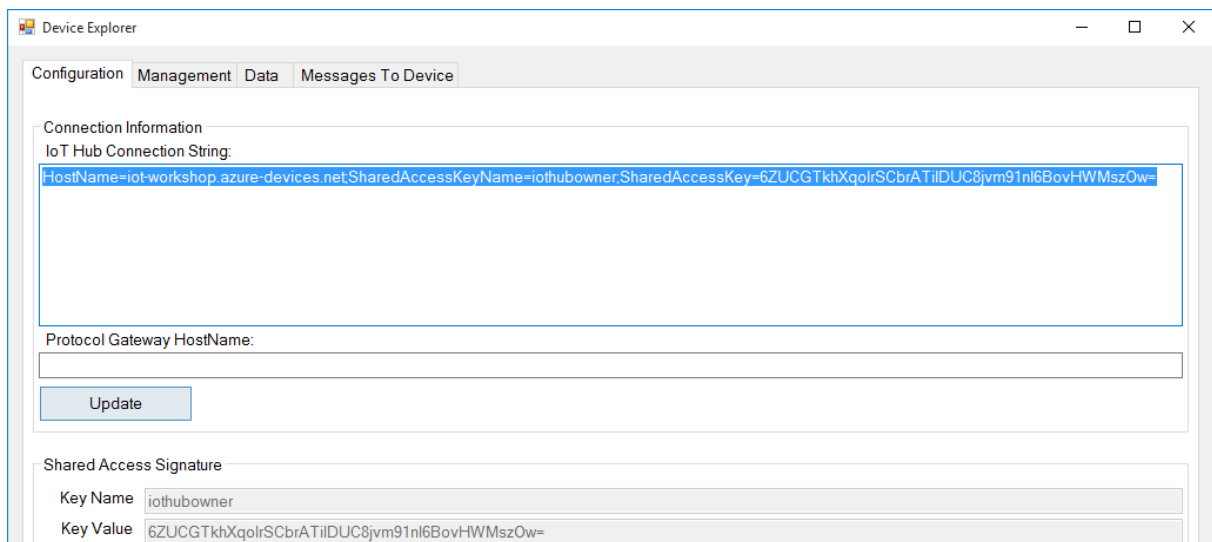
## REGISTERING YOUR DEVICE

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.

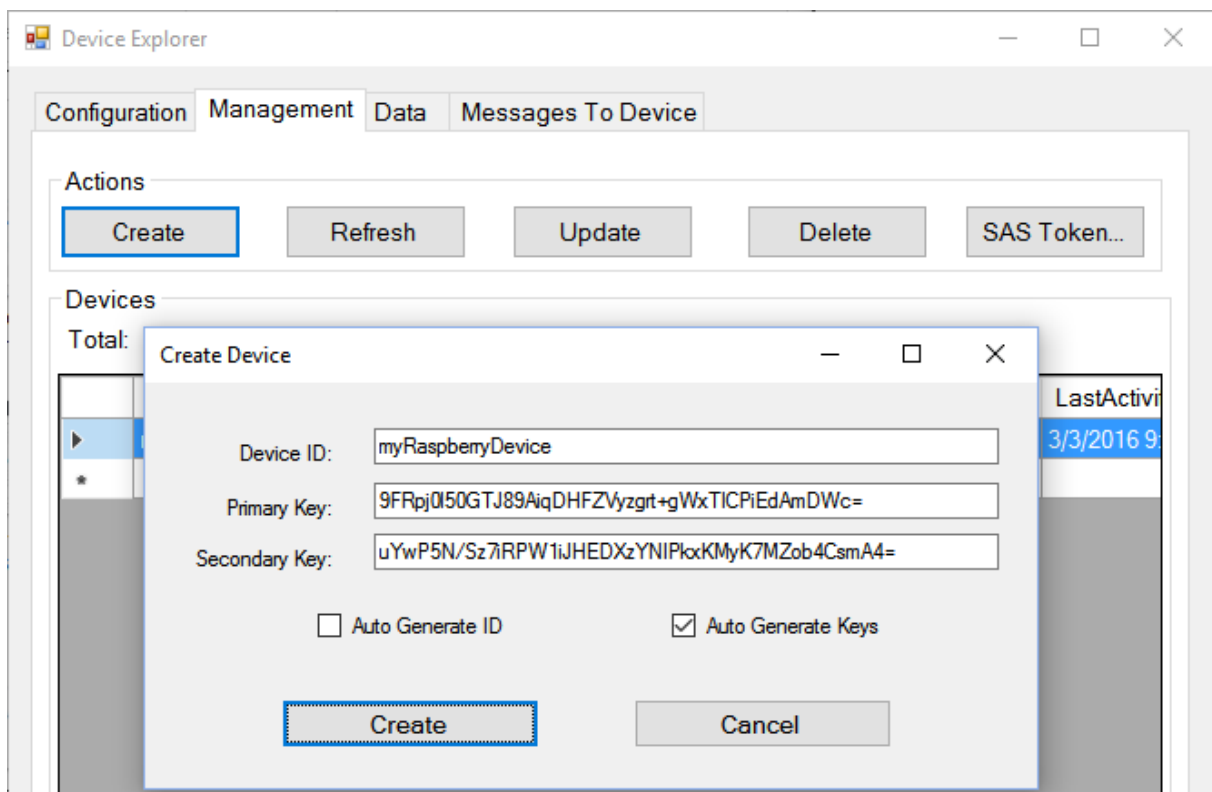
✖ **STEP 1:** Press the Windows key and type “Device Explorer”<sup>6</sup> and run the app.

✖ **STEP 2:** Paste the **IoT Hub Connection String** you previously copied in to the IoT Connection String field and click **Update**.

<sup>6</sup>The Device Explorer is an Open Source sample. In production you would integrate device provisioning in to your solution. See [Get started with Azure IoT Hub for .NET](#).

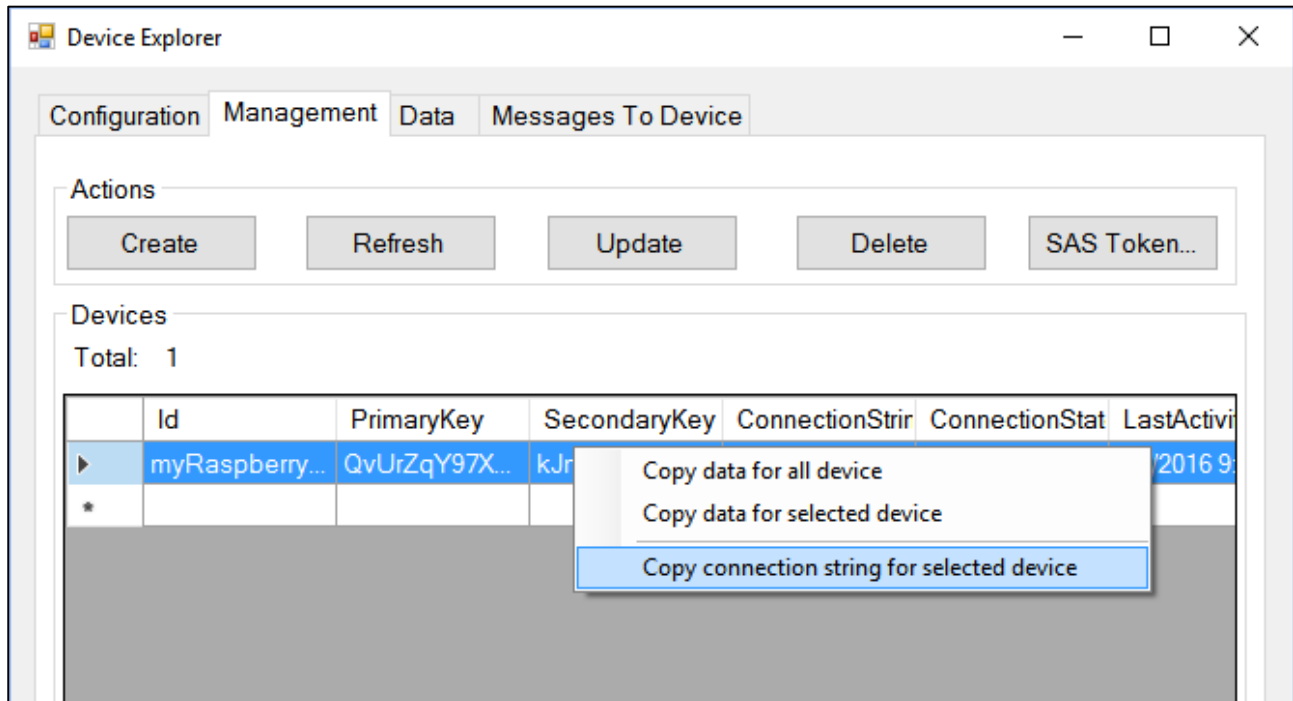


- ✳️ **STEP 2:** Go to the **Management** tab and click on the **Create** button. The Create Device popup will be displayed. Fill the **Device ID** field with a new id for your device. For example, MyRPi01, then click on Create.



- ✳️ **STEP 3:** 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.



---

## EXPERIMENT 6: STREAMING TELEMETRY DATA TO AZURE IOT HUB

✖ **STEP 1:** Back in Visual Studio paste the Connection String in to the highlighted area in the StartUpTask.cs file in the Run method.

```
public async void Run(IBackgroundTaskInstance taskInstance)
{
    DeviceClient deviceClient = DeviceClient.CreateFromConnectionString("Connection String");

    Expand to view variable initialisation

    #region Code snippets to go between the #region and #endregion tags

    while (true)
    {
        var level = hat.GetLightLevel() * 100;

        if (level > LIGHT_THRESHOLD)
        {
            hat.D2.Color = FEZHAT.Color.Blue;
        }
        else
        {
            hat.D2.Color = FEZHAT.Color.Red;
        }

        await Task.Delay(500);
    }

    #endregion
}
```

✖ **STEP 2:** Type the following code in the **Publish** method between the **#region Publish to Azure IoT Hub** tags **OR** using a code snippet type **lab6** and press Tab twice.

```
try // Exception handling if problem streaming telemetry to Azure IoT Hub
{
    hat.D3.Color = publishColor; // turn on publish indicator LED

    var temperature = hat.GetTemperature(); // read temperature from the FEZ HAT
    var light = hat.GetLightLevel() * 100; // read light level from the FEZ HAT
    var json = telemetry.ToJson(temperature, light, 0, 0); //serialise to JSON

    var content = new Message(json);
    await deviceClient.SendEventAsync(content); //Send telemetry data to IoT Hub
}
catch { telemetry.Exceptions++; }
finally { hat.D3.TurnOff(); }
```

✉ **STEP 3:** Your completed Publish method should look like this.

```
async void Publish()
{
    #region Publish to Azure IoT Hub

    try // Exception handling if problem streaming telemetry to Azure IoT Hub
    {
        hat.D3.Color = publishColor; // turn on publish indicator LED

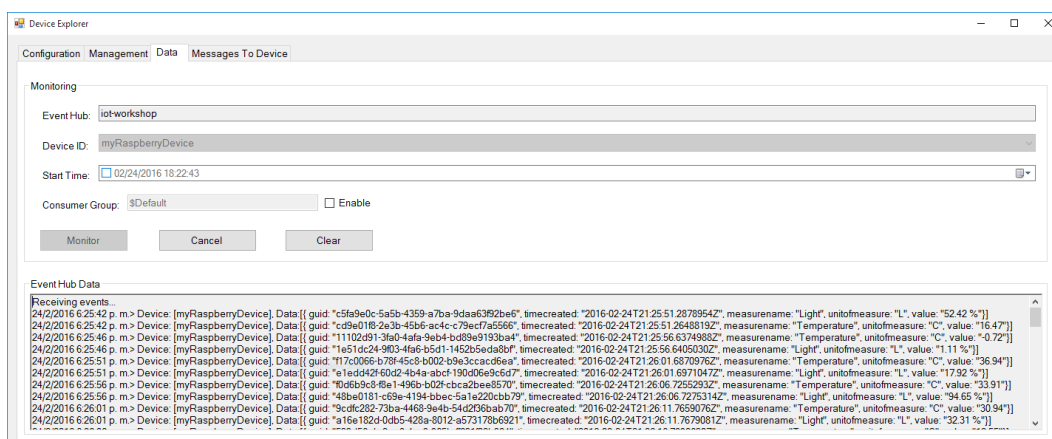
        var temperature = hat.GetTemperature(); // read temperature from the FEZ HAT
        var light = hat.GetLightLevel() * 100; // read light level from the FEZ HAT
        var json = telemetry.ToJson(temperature, light, 0, 0); //serialise to JSON

        var content = new Message(json);
        await deviceClient.SendEventAsync(content); //Send telemetry data to IoT Hub
    }
    catch { telemetry.Exceptions++; }
    finally { hat.D3.TurnOff(); }

    #endregion
}
```

✉ **STEP 4:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start without Debugging** or from the keyboard press **Ctrl+F5** and wait for the solution to deploy.

✉ **STEP 5:** Press the Windows key and type “Device Explorer” and run. Navigate to the Data tab and select your device from the dropdown and click on Monitor.



**Note:** If you navigate back to your IoT Hub blade in the Azure Portal, it may take a couple minutes before the message count is updated to reflect the device activity under **Usage**.

## EXPERIMENT 7: CONSUMING THE IOT HUB DATA

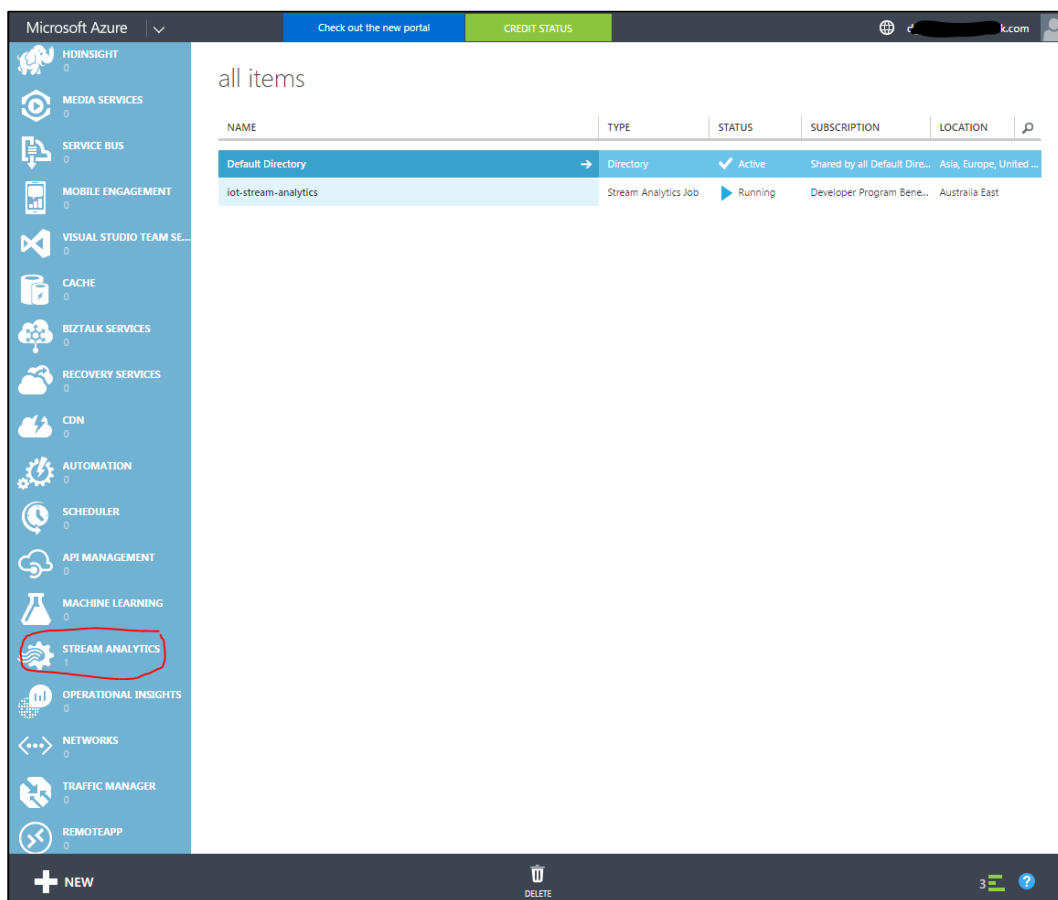
You have used the Device Explorer to view data streamed to the Azure IoT Hub. However, there are many ways to gain insight from the data including Stream Analytics, Power Bi, [Azure IoT Suite Remote Monitoring](#), and of course your own custom solution.

In the following section you will use Azure Stream Analytics in combination with Microsoft Power BI to consume the data and to generate meaningful reports.

### 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 for that job must be provided. As the Raspberry devices are sending information to an IoT Hub, it will be set as the input for the job.

✕ **STEP 1:** From the classic [Azure management portal](#) (<https://manage.windowsazure.com>), select the **Stream Analytics** service.





☒ **STEP 2:** Click "Create a new Stream Analytics job" if this is your first job, or click the plus sign to add an additional job.

☒ **STEP 3:** Complete the form

- **JOB NAME:** iot-stream-analytics or something similar
- **REGION:** Suggest selecting the region closest to your IoT Hub
- **REGIONAL MONITORING STORAGE ACCOUNT:** Likely "Create new storage account"
- **NEW STORAGE ACCOUNT NAME:** Needs to be a globally unique name.

☒ **STEP 4:** Click "CREATE STREAM ANALYTICS JOB."

## CONFIGURE STREAM ANALYTICS

☒ **STEP 1:** Once the Stream Analytics job is created click on it to configure.

☒ **STEP 2:** As you can see, the Start button is disabled since the job is not configured yet. To set the job input click on the **INPUTS** tab and then in the **Add an input** button.

☒ **STEP 3:** In the **Add an input to your job** popup, select the **Data Stream** option and click **Next**. In the following step, select the option **IoT Hub** and click **Next**. Lastly, in the **IoT Hub Settings** screen, provide the following information:

- **Input Alias:** TelemetryHub
- **Subscription:** Use IoT Hub from Current Subscription (you can use an Event Hub from another subscription too by selecting the other option)
- **Choose an IoT Hub:** Choose the IoT hub you previously created
- **IoT Hub Shared Access Policy Name:** iothubowner
- **IoT Hub Consumer Group:** powerbi

ADD AN INPUT

### Add an input to your job

Each job requires at least one data stream input. Adding reference data input is optional.

☒ **Data stream**  
A continuous sequence of data or events to be consumed and transformed by a Stream Analytics job.

☐ **Reference data**  
Auxiliary data used for...

ADD A DATA STREAM

### Add a data stream to your job

☐ Event Hub ?

☐ Blob storage ?

☒ **IoT Hub** PREVIEW

Missing an input source?

### ADD AN IOT HUB

### IoT Hub settings

INPUT ALIAS ?  
TelemetryHub ✓

SUBSCRIPTION  
Use IoT Hub from Current Subscription ▼

CHOOSE AN IOT HUB ?  
sample-iot (eastus) ▼

IOT HUB SHARED ACCESS POLICY NAME ?  
iothubowner ▼

IOT HUB CONSUMER GROUP ?  
powerbi ▼

1 2 4

✕ **STEP 4:** Click **Next**, and then **Complete**, leave the Serialization settings as they are.

## STREAM ANALYTICS OUTPUT SETUP

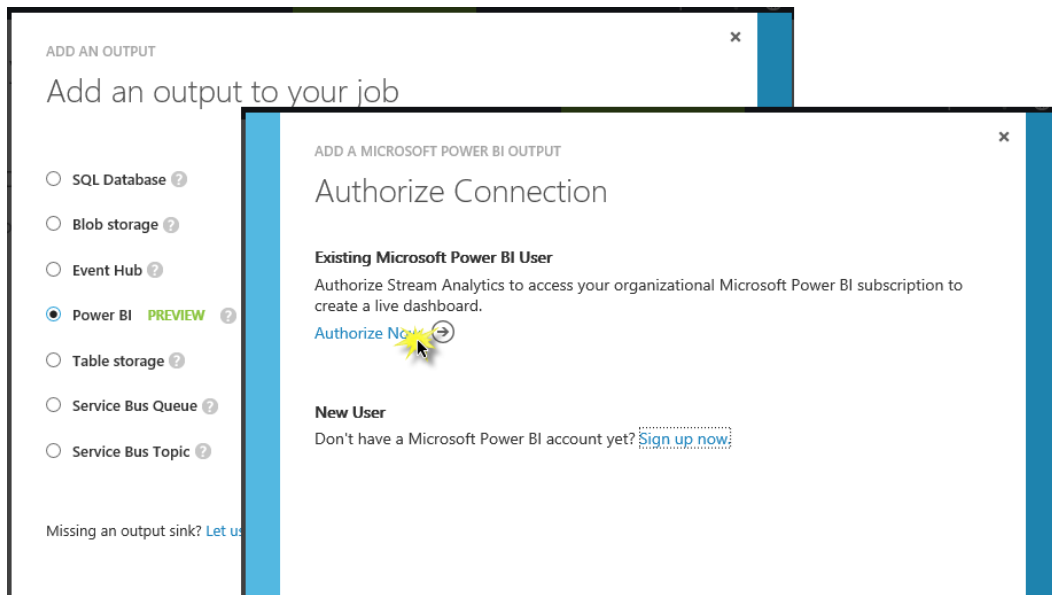
The output of the Stream Analytics job will be Power BI.

✕ **STEP 1:** To set up the output, go to the Stream Analytics Job's **OUTPUTS** tab, and click the **ADD AN OUTPUT** link.

✕ **STEP 2:** In the **Add an output to your job** popup, select the **POWER BI** option and the click the **Next button**.

✕ **STEP 3:** In the following screen you will setup the credentials of your Power BI account in order to allow the job to connect and send data to it. Click the **Authorize Now** link.

✕ **STEP 4:** You will be redirected to the Microsoft login page.



✕ **STEP 5:** Enter your Power BI account email and password and click **Continue**. If the authorization is successful, you will be redirected back to the **Microsoft Power BI Settings** screen.

✕ **STEP 6:** In this screen you will enter the following information:

- ✕ **Output Alias:** PowerBI
- ✕ **Dataset Name:** Raspberry
- ✕ **Table Name:** Telemetry
- ✕ **Group Name:** My Workspace

✕ **STEP 7:** Click the checkmark button to create the output.

## STREAM ANALYTICS QUERY CONFIGURATION

Now that the job's inputs and 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.

✉ **STEP 1:** Go to the Stream Analytics Job **QUERY** tab and replace the query with the following query:

```
SELECT
    iothub.connectiondeviceid deviceid,
    Geo AS GeoLocation,
    Max(DateAdd(Hour, 10, EventEnqueuedUtcTime)) AS TimeCreated, -- AU EST UTC + 10
    Avg(Celsius) AS Temperature,
    AVG(Humidity) AS Humidity,
    AVG(Light) AS Light,
    AVG(HPa) AS AirPressure
INTO
    [PowerBI]
FROM
    [TelemetryHUB] TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY
    iothub.connectiondeviceid, Geo,
    TumblingWindow(Second, 30)
```

The query takes the data from the input (using the alias defined when the input was created **TelemetryHUB**) and inserts into the output (**PowerBI**, the alias of the output) after grouping it using 30 seconds chunks.

✉ **STEP 2:** Click on the **SAVE** button and **YES** in the confirmation dialog.

## STARTING THE STREAM ANALYTICS JOB

Now that the job is configured, the **START** button is enabled. Click the **START**<sup>7</sup> and then select the **JOB START TIME** option in the **START OUTPUT** popup. After clicking **OK** the job will be start, it will take a couple of minutes for the service to be operational.

Once the job starts and it is processing data it will create the Power BI datasource associated with the given subscription.

---

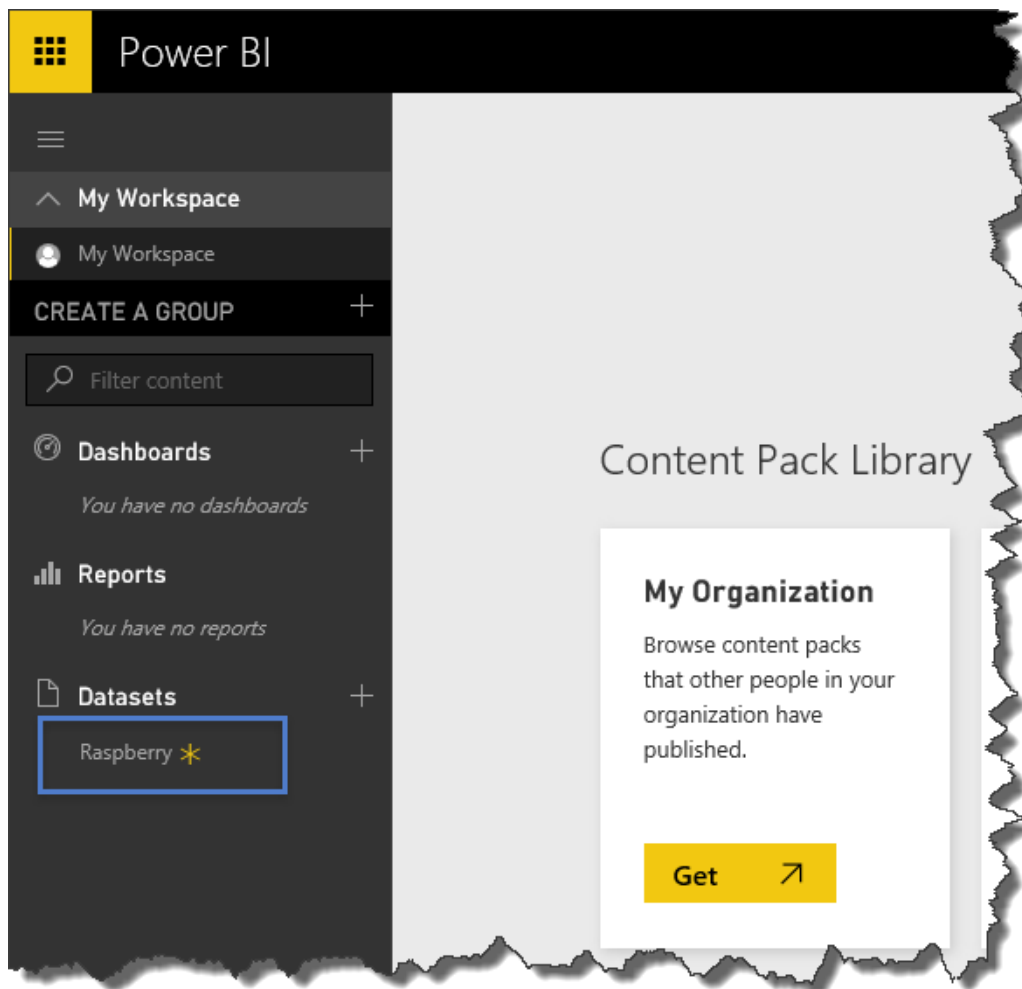
<sup>7</sup>Starting a Stream Analytics job will start to consume your Azure Credits.

---

## EXPERIMENT 8: SETTING UP THE POWER BI DASHBOARD

✉ **STEP 1:** Navigate to Power Bi ([www.powerbi.com](https://www.powerbi.com)) and authenticate. Click the Hamburger to expand the navigation pane.

The Steam Analytics<sup>8</sup> job needs to run for a few minutes before it appears in the navigation pane.



✉ **STEP 2:** Click on the **Raspberry** datasource to start defining the report.

---

<sup>8</sup>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 Raspberry dataset then check the Universal App is running on the Raspberry Pi and it is streaming data to Azure. 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

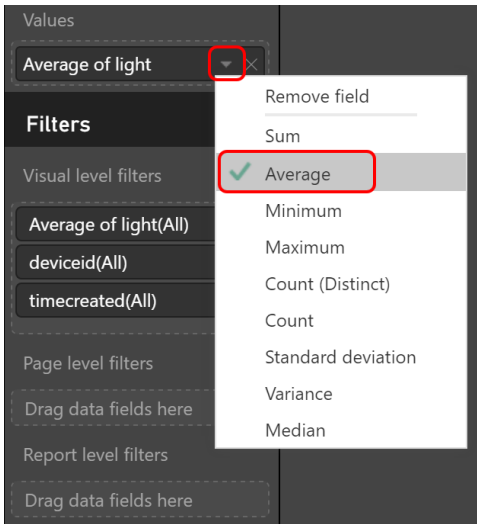
The Report designer will be opened showing the list of fields available for the selected datasource and the different visualizations supported by the tool.

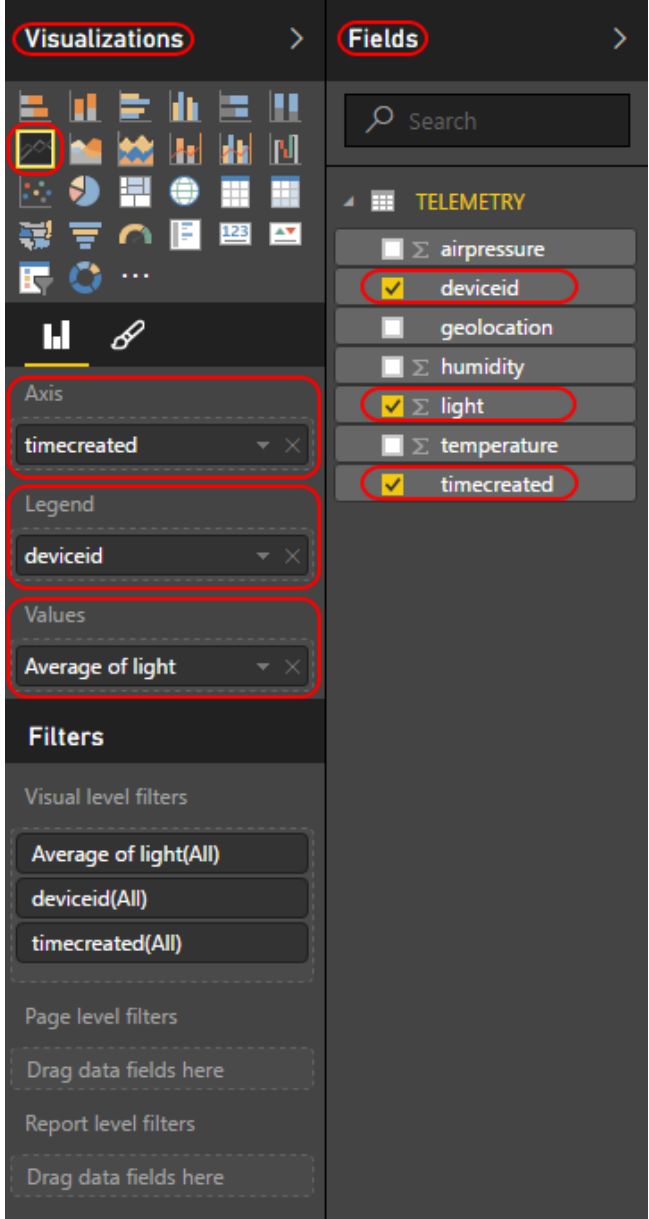
✕ **STEP 1:** Select Line Chart from the Visualizations

✕ **STEP 2:** Drag and drop the following fields from the Fields section

- 1) deviceid -> Legend
- 2) light -> Values
- 3) timecreated -> Axis

✕ **STEP 3:** Select **Average of light** from the Values dropdown menu.

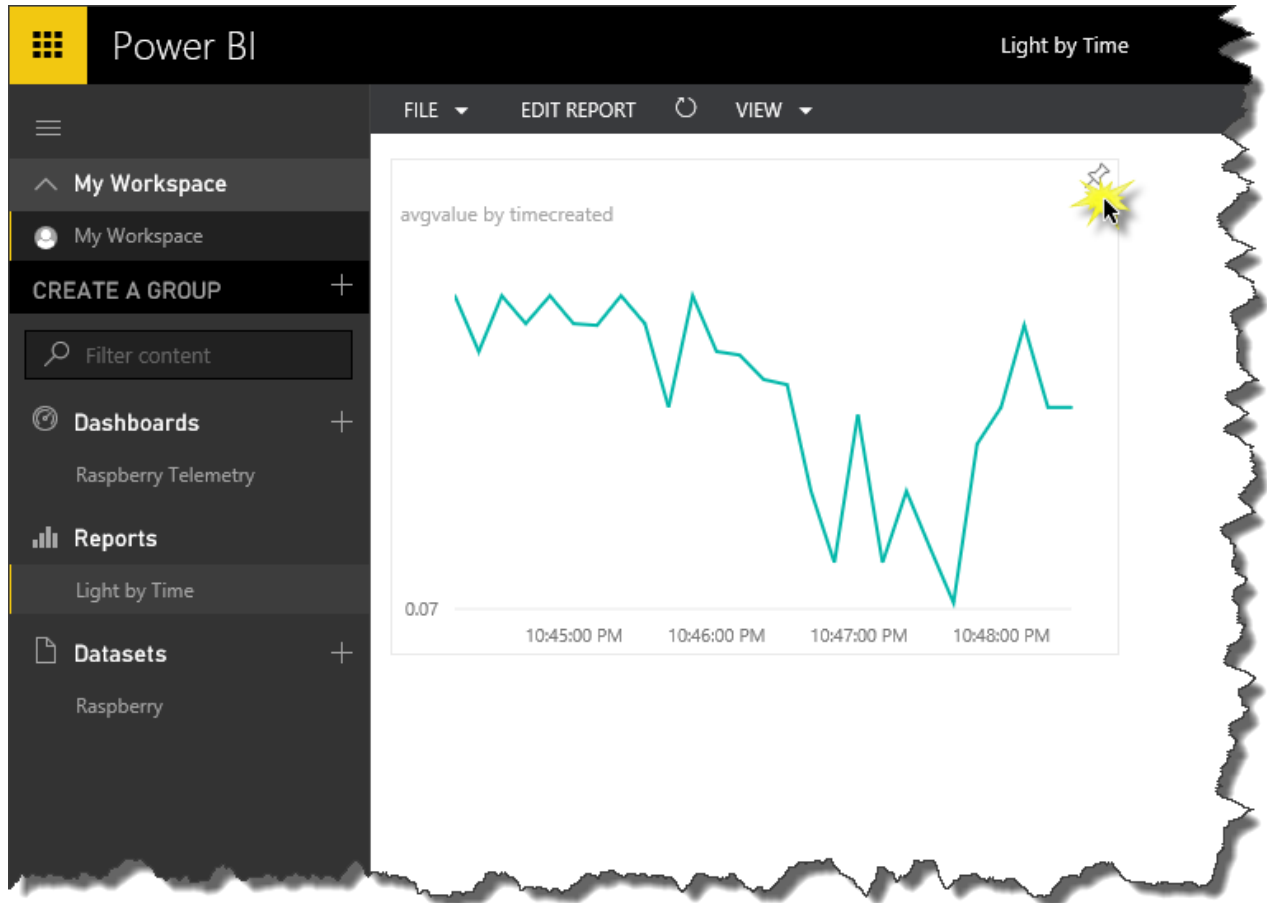




✕ **STEP 4:** Click the **SAVE** button and set LIGHT BY TIME as the name for the report.

✕ **STEP 5:** Now create a new Dashboard, and pin this report to it. Click the plus sign (+) next to the **Dashboards** section to create a new dashboard and name it Raspberry Telemetry.

✕ **STEP 6:** Now, go back to your report and click the pin icon to add the report to the newly created dashboard.



## TEMPERATURE POWER BI CHART

Create a second chart displaying the average Temperature.

☒ **STEP 1:** Click on the **Raspberry** datasource to create a new report.

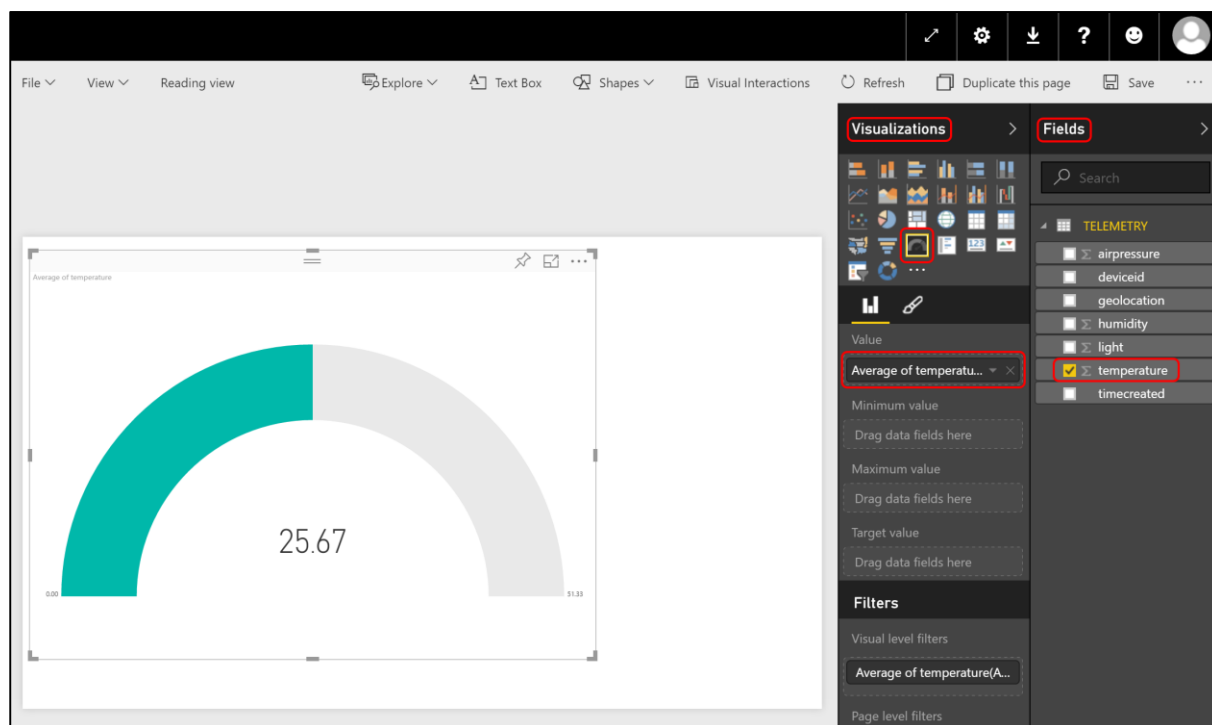
☒ **STEP 2:** From the Visualizations click on the **Gauge Chart**.

☒ **STEP 3:** Drag the **temperature** field to the Value Data Field.

☒ **STEP 4:** Change the **Value** from **Sum** to **Average**.

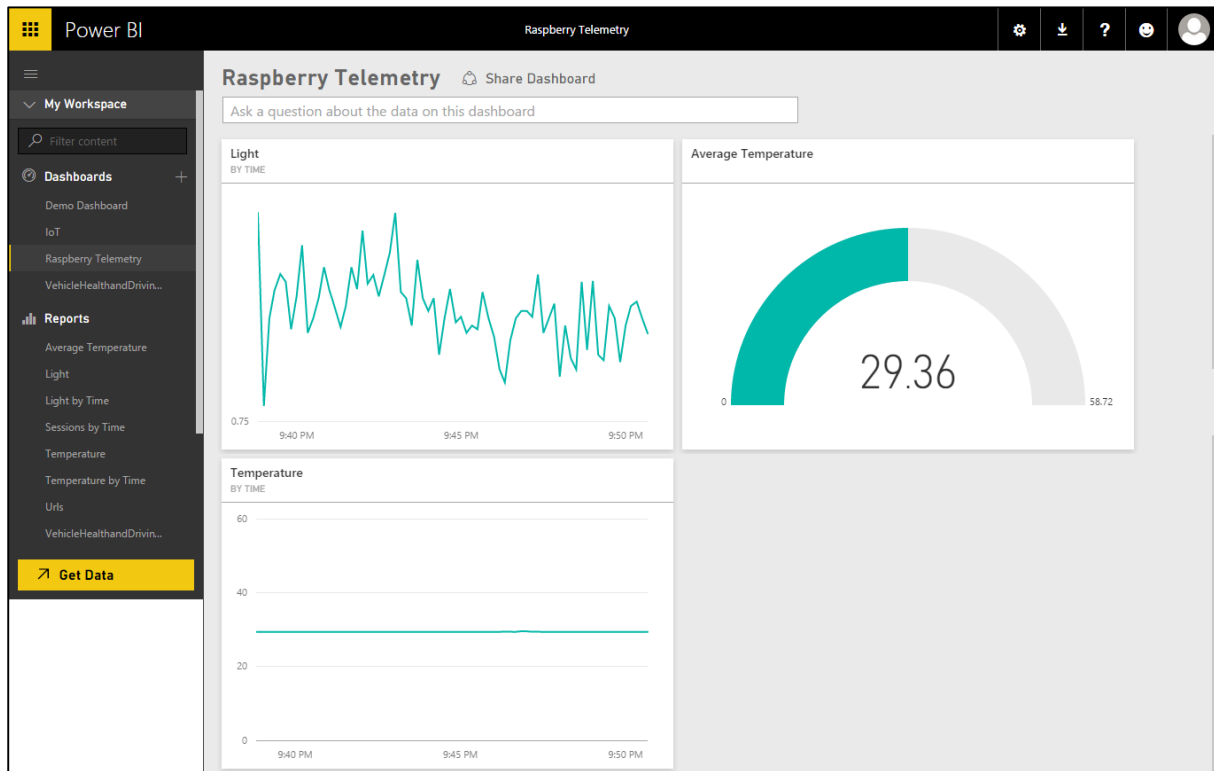
☒ **STEP 5:** Following the same directions, create a TEMPERATURE report and add it to the dashboard.

☒ **STEP 6:** Lastly, edit the reports name in the dashboard by clicking the pencil icon next to each report.





After renaming both reports you will get a dashboard similar to the one in the following screenshot, which will be automatically refreshed as new data arrives.



☒ **STEP 7:** Try the Power Bi apps available in the iOS App store, Google Play and Windows Stores.

---

## EXPERIMENT 9: CONTROLLING A DEVICE FROM AZURE IOT HUB

Azure IoT Hub is a service that enables reliable and secure bi-directional communications<sup>9</sup> between millions of IoT devices and an application back end.

In this experiment we will send cloud-to-device messages to your device to command it to change the colour of one of the FEZ HAT LEDs. For the experiment Device Explorer will serve as the back end.

☒ **STEP 1:** Type the following code in the **Commanding\_CommandReceived** method between the **#region IoT Hub Command Support** tags **OR** using a code snippet type **lab9** and press Tab twice.

```
#region IoT Hub Command Support

char cmd = e.Item.Length > 0 ? e.Item.ToUpper()[0] : ' '; // get command character sent from IoT Hub

switch (cmd)
{
    case 'R':
        publishColor = FEZHAT.Color.Red;
        break;
    case 'G':
        publishColor = FEZHAT.Color.Green;
        break;
    case 'B':
        publishColor = FEZHAT.Color.Blue;
        break;
    case 'Y':
        publishColor = FEZHAT.Color.Yellow;
        break;
    case 'M':
        publishColor = FEZHAT.Color.Magenta;
        break;
    default:
        System.Diagnostics.Debug.WriteLine("Unrecognized command: {0}", e.Item);
        break;
}

hat.D3.Color = publishColor;

#endregion
```

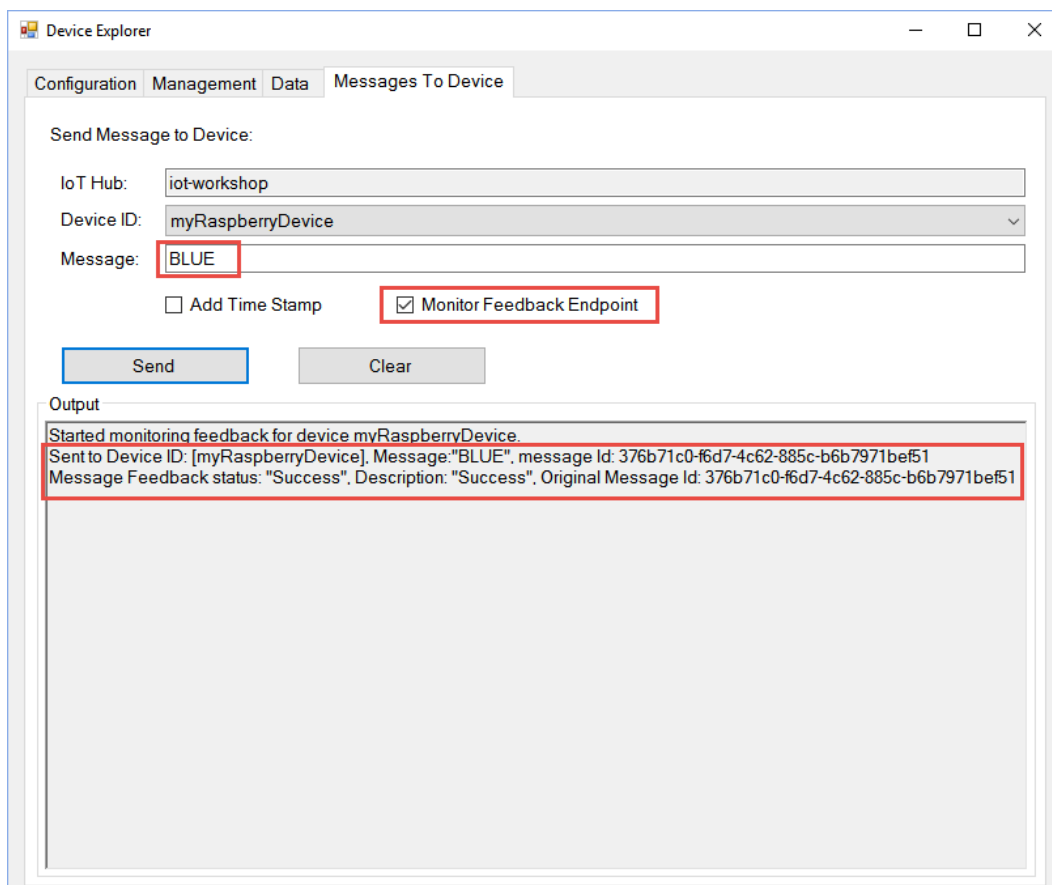
---

<sup>9</sup> Azure IoT Hub supports a number of protocols including [AMQP](#), HTTPS and [MQTT](#).

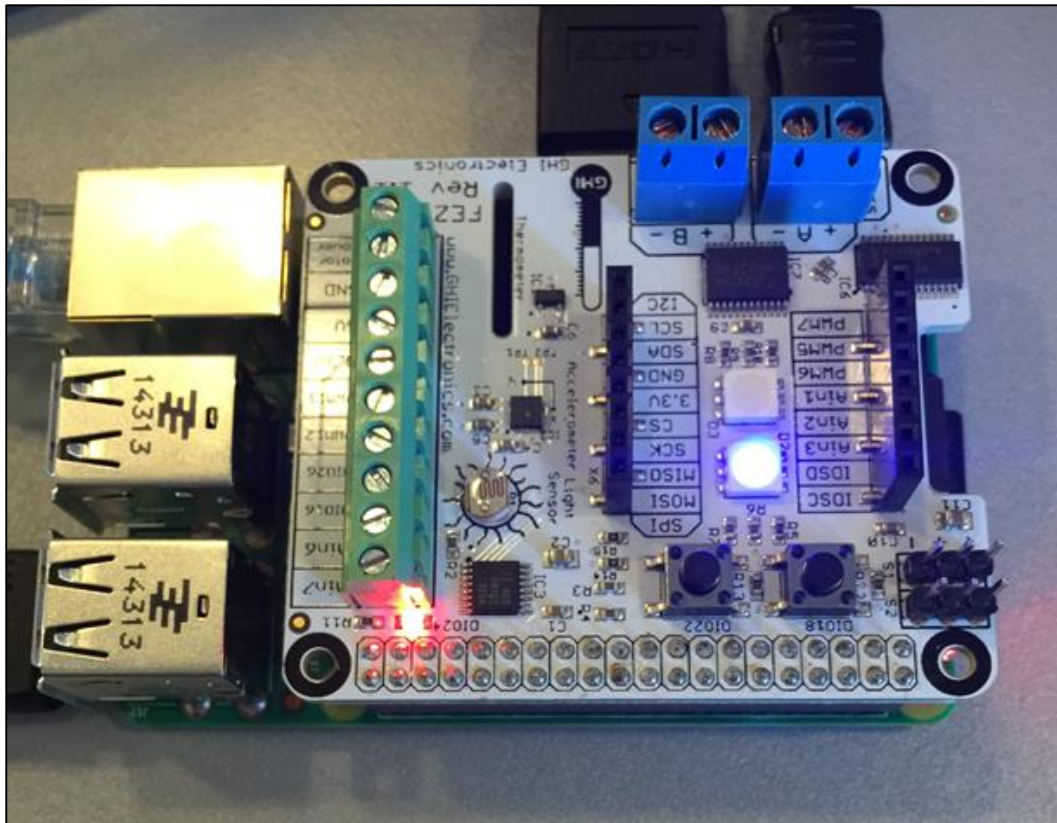
✕ **STEP 2:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start without Debugging** or from the keyboard press **Ctrl+F5** and wait for the solution to deploy.

✕ **STEP 3:** From Device Explorer select the Messages to Device Tab, select your device from the Device ID: dropdown and in the Message field type a colour. Valid colours are Red, Green, Blue or Yellow – or just the first letter of a colour.

✕ **STEP 4:** Enable **Monitor Feedback Endpoint** and click Send.



After a few seconds the message will be processed by the device and the LED will turn on in the colour you selected. The feedback will also be reflected in the Device Explorer screen after a few seconds.



Congratulations, you have finished!

## EVALUATION

Congratulations, you have successfully completed the Maker Den Experience. You have deployed a Universal Windows App to a Raspberry Pi. You have streamed data to Microsoft Azure, ingested telemetry using Azure IoT Hub and visualised data with the Power Bi.

Please complete the following steps before you leave.

☒ **STEP 1:** Close Visual Studio.

All the documentation and software for the Maker Den is available at <http://www.github.com/makerden>

## APPENDIX

### PROVISION AN AZURE ACCOUNT

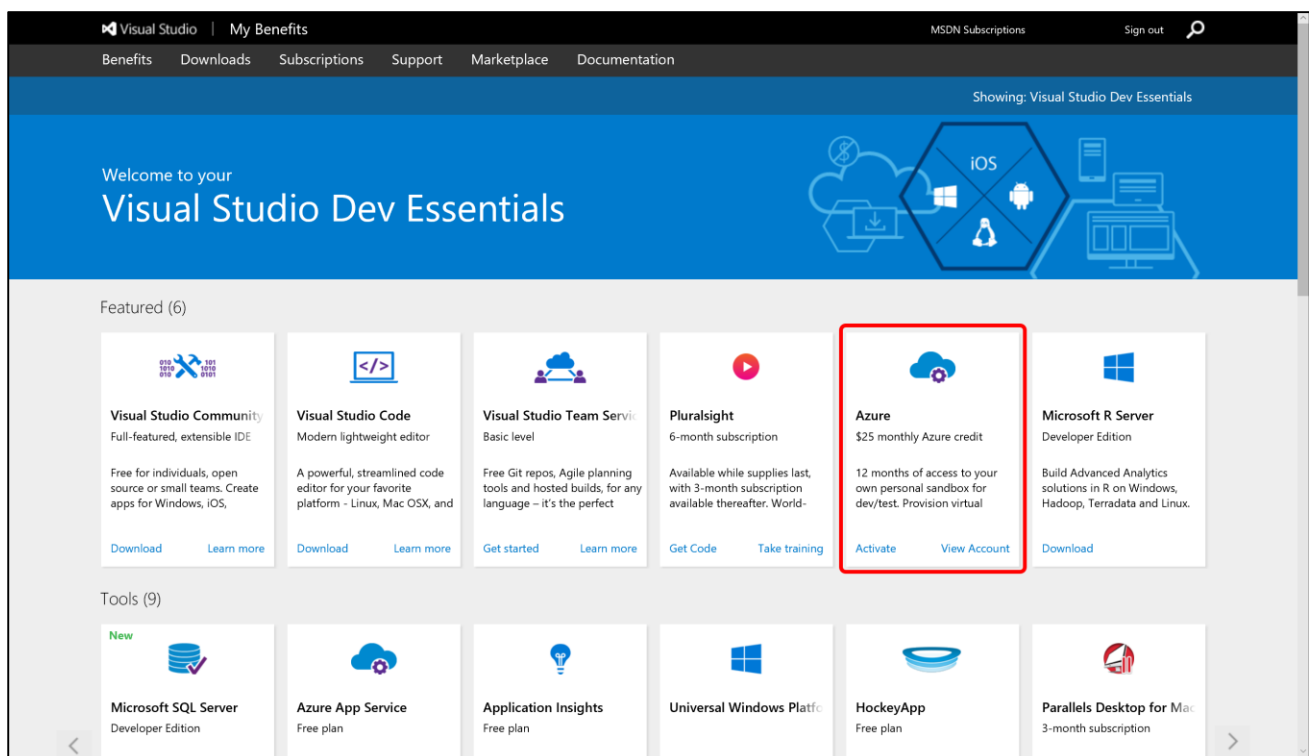
If you don't already have an Azure account, then you will need to provision one.

There are currently two free trail offers – either good for the purposes of the workshop.

- 1) [Visual Studio Dev Essentials](#) Sign up for free. \$25 a month for a year. More slow and steady over an extended period of time.
- 2) [Free one-month trial](#). Sign up for free and get \$200 to spend on all Azure services. Great if you really want to exercise lots of Azure capabilities for a limited period of time.

Valid credit card information is required for identity verification purposes only. Your credit card will not be charged for this offer unless you explicitly remove the spending limit.

If you sign up for Dev Essentials, then be sure to select the Azure offering highlighted below.



## TROUBLESHOOTING

### FORCING A TIME RESYNC

1. From “Windows 10 IoT Core Dashboard”, right mouse click your device and Connect using PowerShell
2. Authenticate
3. At the command prompt, type “w32tm /resync” and press the Enter key to execute.

### LAST BOOT DATE AND TIME

From PowerShell

```
wmic os get lastbootuptime
```

### 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 wlan delete profile *ProfileName*
- netsh wlan add profile Wi-Fi- *ProfileName.xml*
- netsh wlan connect name= *ProfileName*
- netsh interface ipv4 set dns "Wi-Fi" static 192.168.1.1
- netsh interface ipv4 set address "Wi-Fi" static 192.168.1.107 255.255.255.0 192.168.1.1