# TP : 3 Explorations in Dot Net Core 3 for Raspberry Pi

**Objectives :**

1. Create of a free Azure Account
2. Install DOTNET Core SDK and Realize our first Hello World application
3. [Send message from device to Azure IoT Hub](https://www.petecodes.co.uk/explorations-in-dot-net-core-3-0-for-raspberry-pi-part-3/)
4. **Creation of a free Azure Account**

# url 1: <https://azure.microsoft.com/en-in/free/students/>

# url 2 (for help): <https://www.youtube.com/watch?v=tdoflTq9Gc0>

# DotNet Core SDK Installation and Hello World First Application

## **What you’ll need**

* A recent Raspberry Pi – [Go shopping at the Pi Hut.](https://thepihut.com/collections/raspberry-pi)
* You’ll also need to setup a Samba Share, create a folder in your home directory called “**share**” and share that – [Here’s a link to a good guide.](https://www.raspberrypi.org/magpi/samba-file-server/)
* Then you’ll need a SSH Client like PUTtY – [Get it here](https://www.putty.org/).
* Next up you’ll need a Code Editor – [Use Visual Studio Code](https://code.visualstudio.com/)!

I’ll be using a [Raspberry Pi 3B+](https://thepihut.com/collections/raspberry-pi/products/raspberry-pi-3-model-b-plus)… It’s important to note that Dot Net Core will only work on an AMR32v7 processor or above… So that’s a Raspberry Pi 2 and upwards… Unfortunately this rules out the Pi Zero and Pi Zero W as they have V6 processors.

**Installation of .NET Core 3.x on Rasp**

# url 3: <https://www.youtube.com/watch?v=WDlZ3f2xHcc>

## **Verify your installation**

At this stage the setup of Dot Net Core should be done… So time to find out if everything is working… Run the following to see if it’s all ok;

dotnet --info

If all went well, you should see the Dot Net Version Information reported in the terminal window;

Une image contenant texte

Description générée automatiquementDot Net Core Version Information

## **Time to scaffold our app**

Firstly we need to create a directory for our project;

cd share

mkdir ConsoleAPP

cd ConsoleAPP

Now we can scaffold our new console Core App;

dotnet new console

You need now to setup a Samba Share, create a folder in your home directory called “*share*” and share it with win 10 operating system

### **Install Samba**

Samba is available in Raspbian’s standard software repositories. We’re going to update our repository index, make sure our operating system is fully updated, and install Samba using apt-get. Open a Terminal and type:

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install samba samba-common-bin

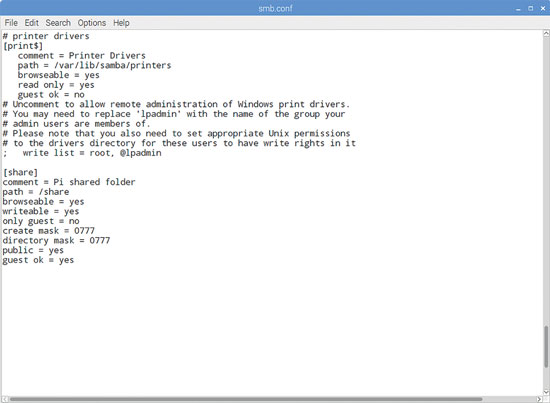
### **Create your shared directory**

We’re going to create a dedicated shared directory on our Pi’s micro SD hard disk. You can put it anywhere, but ours will be at the top level of the root file system.

sudo mkdir -m 1777 /share

This command sets the sticky bit (1) to help prevent the directory from being accidentally deleted and gives everyone read/write/execute (777) permissions on it.

Configure Samba to share your new directory



Samba Config files

Edit Samba’s config files to make the file share visible to the Windows PCs on the network.

sudo nano /etc/samba/smb.conf

In our example, you’ll need to add the following entry:

[share]

Comment = Pi shared folder

Path = /share

Browseable = yes

Writeable = Yes

only guest = no

create mask = 0777

directory mask = 0777

Public = yes

Guest ok = yes

This means that anyone will be able to read, write, and execute files in the share, either by logging in as a Samba user (which we’ll set up below) or as a guest. If you don’t want to allow guest users, omit the guest ok = yes line.

You could also use Samba to share a user’s home directory so they can access it from elsewhere on the network, or to share a larger external hard disk that lives at a fixed mount point. Just create a smb.conf entry for any path you want to share, and it’ll be made available across your network when you restart Samba.

### **Create a user and start Samba**

Before we start the server, you’ll want to set a Samba password - this is not the same as your standard default password (raspberry), but there’s no harm in reusing this if you want to, as this is a low-security, local network project.

sudo smbpasswd -a pi

Then set a password as prompted. Finally, let’s restart Samba:

sudo /etc/init.d/samba restart

From now on, Samba will start automatically whenever you power on your Pi. Once you’ve made sure that you can locate your shared folder on the network, you can safely disconnect the mouse, monitor, and keyboard from your Pi and just leave it running as a headless file server.

### **Find your Pi on the network**

You’ll now be able to find your Raspberry Pi file server (named RASPBERRYPI by default) from any device on your local network. If you’ve left smb.conf’s default settings as they are, it will appear in a Windows network workgroup called WORKGROUP.

Sisable your firewall if you don’t succeed to access your shared folder from win10

## **Are you listening?**

Now we have access to our files, we can make the edits we need…

Basically, before we can test that our “Hello World” style console app is working, we need to allow the app to be accessed from an external PC (That is, unless you’re happy to test directly on the Pi).

We can accomplish this by adding an extra line to the “program.cs” file in the root of our project.

First, let’s open our project in Visual Studio Code… if you’re on Windows and you’ve followed along with the steps above, you can enter your “rpiconsole” folder, find some blank space and right click. The menu shown will include a “Open with Code” item;

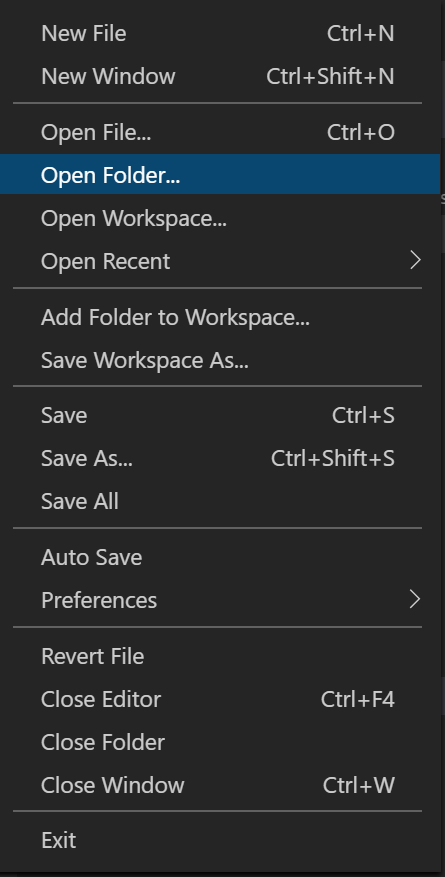
Une image contenant texte

Description générée automatiquement

Open with Code Menu Item

Hitting “Open with Code” will open the whole folder in Visual Studio Code, allowing us easy access to all the files.

Alternatively, you can open VS Code yourself and choose the “Open Folder” option from the File Menu;



Open Folder Menu Item

Once you have your project loaded in VS Code, then open the “*Program.cs*” file from the Explorer on the left;

## **Run Forest Run!**

We’re now ready to run our Web App. Return to your SSH client and make sure you’re in your application directory.

Then run the following command to build your console Core application;

dotnet build

This may take some time on a Pi (2 or possibly even 3 minutes), as there’s quite a lot to do for this diminutive device! So just leave it be and wait for the magic to complete. It may look like nothing is happening for a while, so just be patient!

Une image contenant texte

Description générée automatiquement

Dot NET Build Command

You can see in the above image that this took 2 minutes and 13 seconds on my Pi 3B+.

Next up, we can run our app… YEY! Still in the SSH client, run the following command to spin up the server;

dotnet run

This will take slightly less time than building, so you should see some results pretty quickly;

Une image contenant texte

Description générée automatiquementDot NET Run Command

Once you get to this point, you will see a helloWord message displayed on the screen

1. [***Receiving Azure IoT Hub Messages***](https://www.petecodes.co.uk/explorations-in-dot-net-core-3-0-for-raspberry-pi-part-4/)

## **Putting the “I” in IoT**

If you’ve followed along with the previous sections, you’ll already have installed the Dot Net Core 3.0 binaries and runtimes, and have a console app running.

Our next step is to hook our Raspberry Pi up to an Azure IoT hub to add some internet goodness.

To do this, we’ll be adding a reference to a Nuget package that allows us to communicate with the Azure IoT Platform, and the IoT Hub in particular.

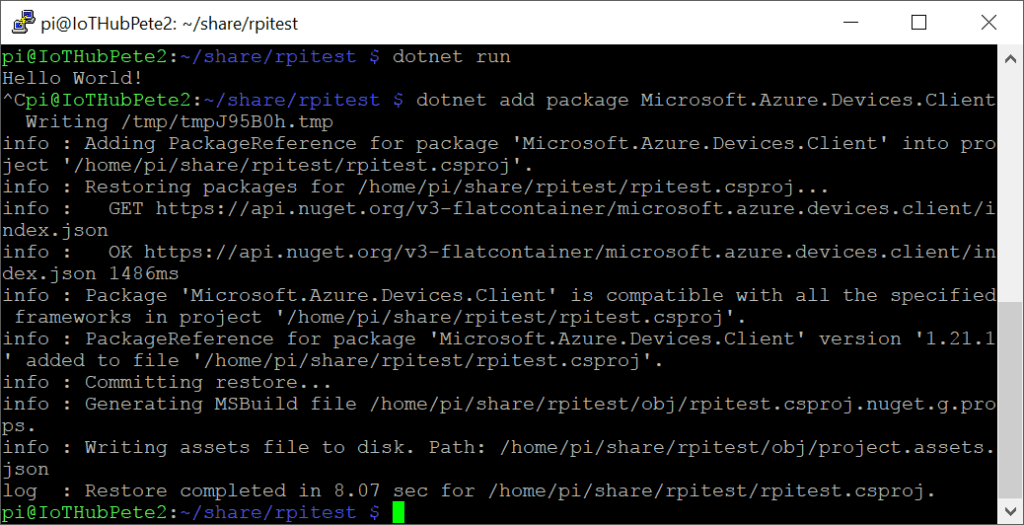
We’ll then modify our console applications to allow us to send the message.

Finally we’ll add a subroutine that will create our message and send it across to the IoT Hub.

## **The return of the Nug(g)et**

The first thing we need to is add a reference to the [Microsoft.Azure.Devices.Client Nuget Package](https://www.nuget.org/packages/Microsoft.Azure.Devices.Client/" \t "_blank) to our project, we can add this by running;

dotnet add package Microsoft.Azure.Devices.Client



Add Azure Devices Package

## **Let’s see your references**

Switching back to our program.cs file, we now need to add a reference Microsoft.Azure.Devices.Client to allow us to communicate with the IoT Hub, We’ll also need a reference to System.Text which will allow us to encode our message to the IoT Hub correctly.

Add the following to the references at the top of the file;

using Microsoft.Azure.Devices.Client;

using System.Text;

### **Global Variables**

We’re going to need a couple of global variables. We’ll need a string to hold our IoT Hub Connection String. We’ll also need a reference to the object that we’ll be using to communicate with the Azure IoT Hub.

Add the following two lines above the void main;

private static readonly string connectionString = "[Connnection String]";

private static DeviceClient deviceClient;

### **Local Variables**

We now need to instantiate our deviceClient object. Add the following beneath the line where we instantiate the GpioController;

deviceClient = DeviceClient.CreateFromConnectionString(connectionString);

## **Message in a bottle**

We’ve now got the basic framework written, it’s time to create the subroutine that will send a message the IoTHub.

After the closing brace to our “*main*” sub create a new private async void subroutine called ” SendDeviceToCloudMessageAsync;

private static async void SendDeviceToCloudMessageAsync(string MonMessage)

{

}

We’re creating an Async Void here because the function we’re using to send a message to the IoT Hub will be an Async one also.

The IoT Hub expects messages to be encoded correctly. In this case, we’ll be encoding our message as a series of ASCII Bytes. Add the following line below the new string to encode the message;

Message message = new Message(Encoding.ASCII.GetBytes(MonMessage));

We can now add some properties to our message. It’s possible to then use these properties to filter messages in Azure later on if required;

message.Properties.Add("buttonEvent", "true");

At this stage we have everything we need to send a message to the IoT Hub;

await deviceClient.SendEventAsync(message);

Here’s we’re using an Async call – SendEventAsync to send our encoded message to the IoT Hub.

So we’re able to see in our console that we’re successfully sending the message let’s add;

Console.WriteLine("Sending Message {0}", messageString);

The SendDeviceToCloudMessageAsync sub should now look like;

private static async void SendDeviceToCloudMessageAsync (String monMessage)

{

var messageString = monMessage;

Message message = new Message(Encoding.ASCII.GetBytes(messageString));

message.Properties.Add("buttonEvent", "true");

await deviceClient.SendEventAsync(message);

Console.WriteLine("Sending Message {0}", messageString);

}

## **The Finished Code**

Your finished code should now look like;

using System;

using System.Threading;

using Microsoft.Azure.Devices.Client;

using System.Text;

namespace HelloWordApplication

{

class Program

{

private static readonly string connectionString = "[iot Azure connection string]”

private static DeviceClient deviceClient;

static void Main(string[] args)

{

deviceClient = DeviceClient.CreateFromConnectionString(connectionString);

Console.WriteLine("Hello World Modified From VS Code!");

//String message;

while (true)

{

Console.WriteLine("Tappez votre message");

String monMessage = Console.ReadLine();

SendDeviceToCloudMessageAsync(monMessage);

Thread.Sleep(2000);

}

}

private static async void SendDeviceToCloudMessageAsync(String monMessage)

{

Message message = new Message(Encoding.ASCII.GetBytes(monMessage));

message.Properties.Add("buttonEvent", "true");

await deviceClient.SendEventAsync(message);

Console.WriteLine("Sending Message {0}", messageString);

}

}

}

## **Through the Portal**

Now we have some software, we need to create an IoT Hub. Head over to [portal.azure.com](https://portal.azure.com/) and sign in.

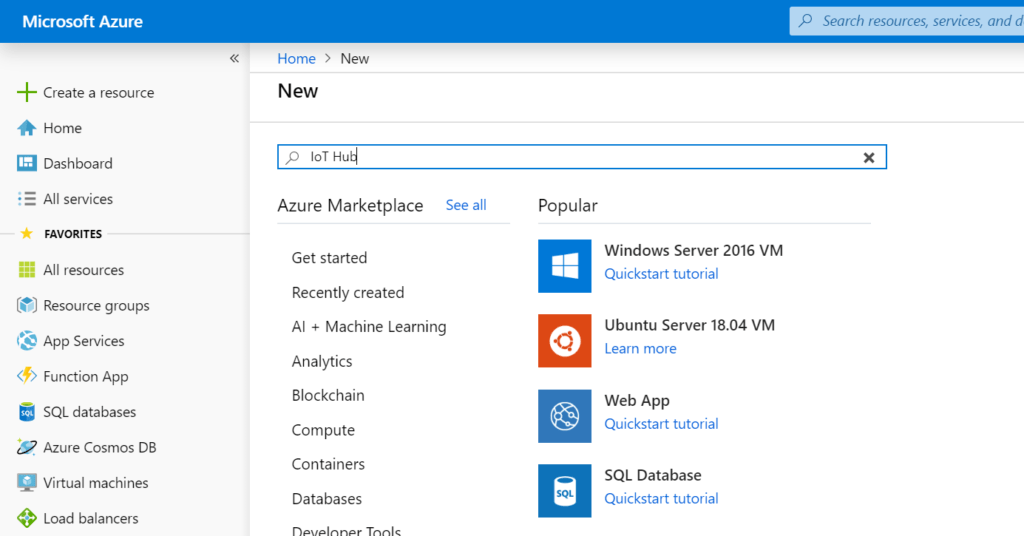
## **Create an IoT Hub**

Once your signed in click the Create Resource button at the top left;



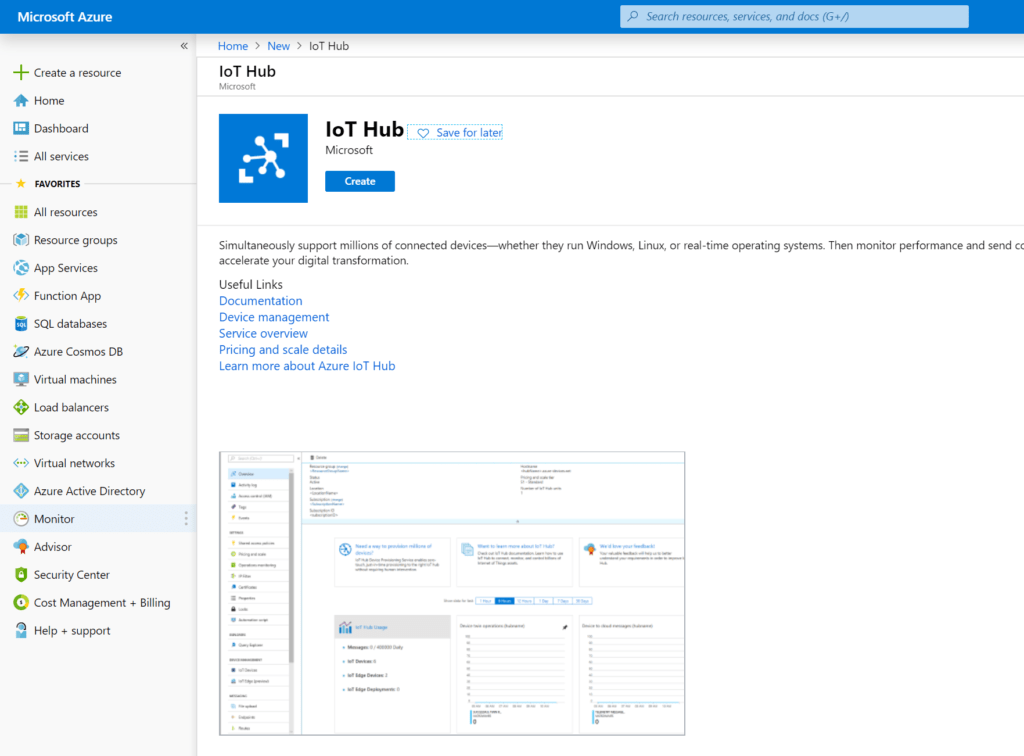
Azure Create Resource

In the search box that appears, enter “IoT Hub”;



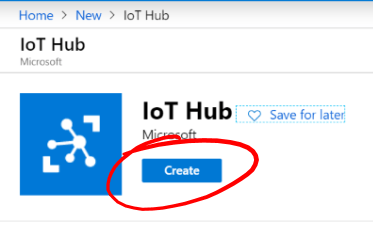
Azure Create a Resource

* **Pressing enter will show the IoT Hub resource;**



Azure IoT Hub

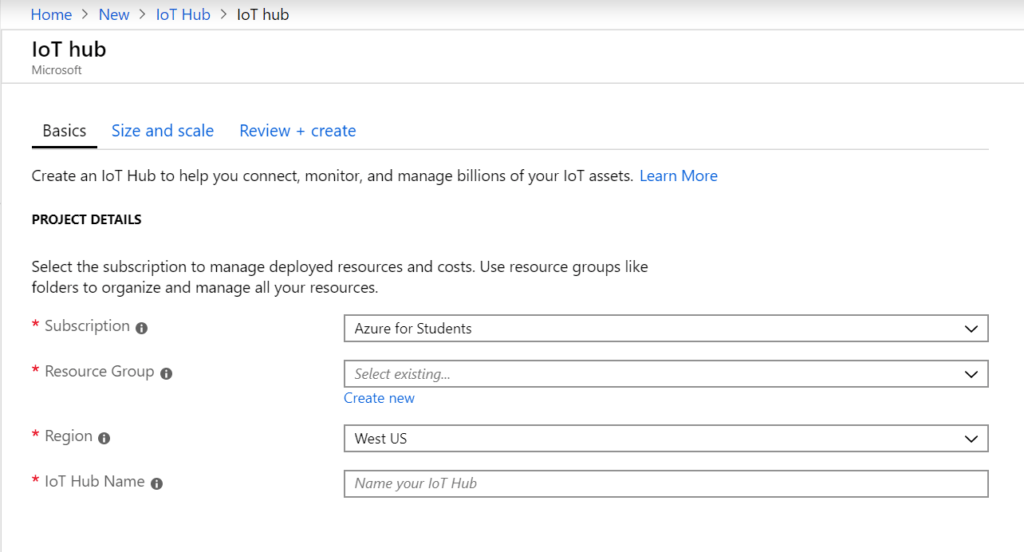
* **Create the IoT Hub by hitting the “Create” button;**



Create IoT Hub Button

* **Iot Hub Options**

You’ll be shown the basic page of the process to create an IoT Hub;

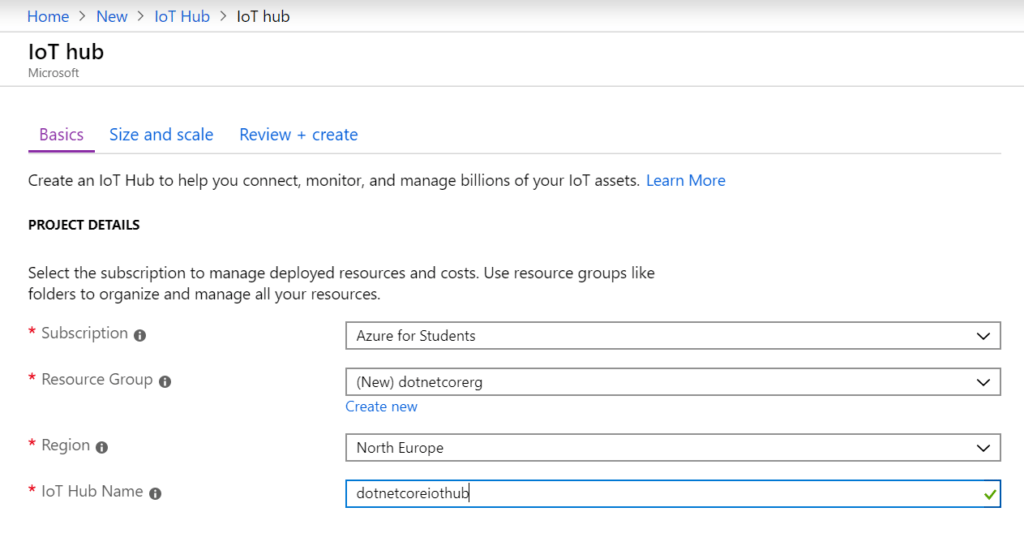


Create IoT Hub – Basic

Firstly choose your subscription. Next you should create a new Resource Group using the blue “Create new” link under the Resources Group section. Name this something like “dotnetcorerg”

Next choose your resource group. I chose North Europe.

Finally enter a name for your IoT Hub, for instance, “dotnetcoreiothub”.

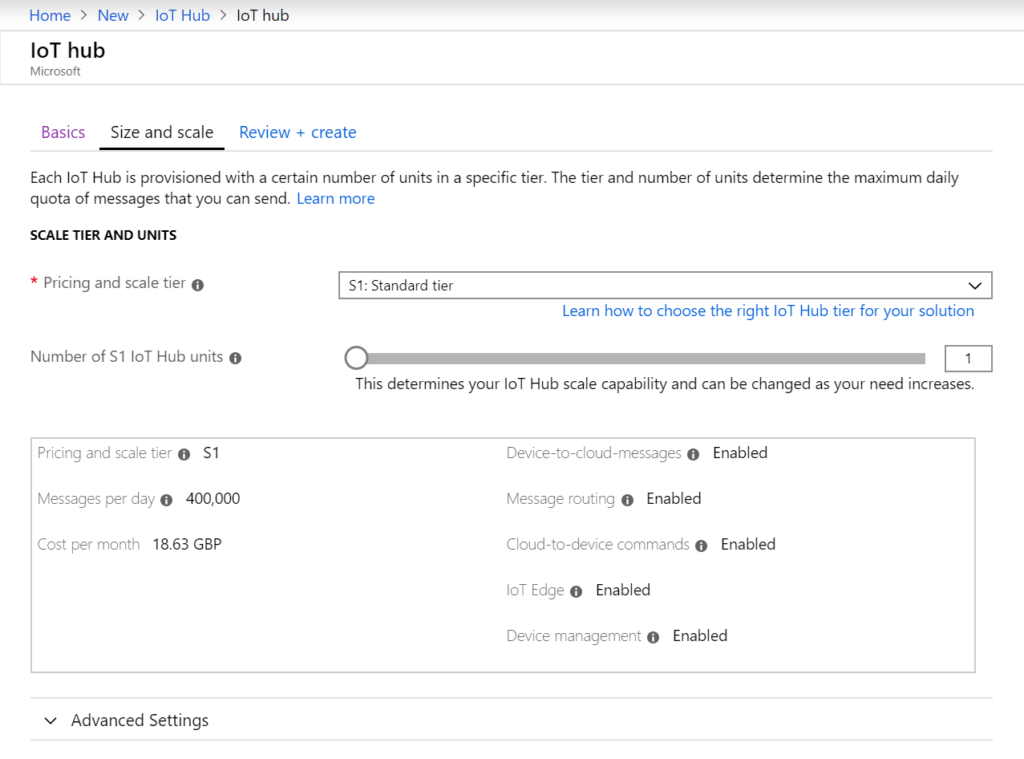


IoT Hub Setup

* **Next press the “Next: Size and Scale >” Button;**

Size and Scale Button

You’ll now see the Size and Scale settings page;

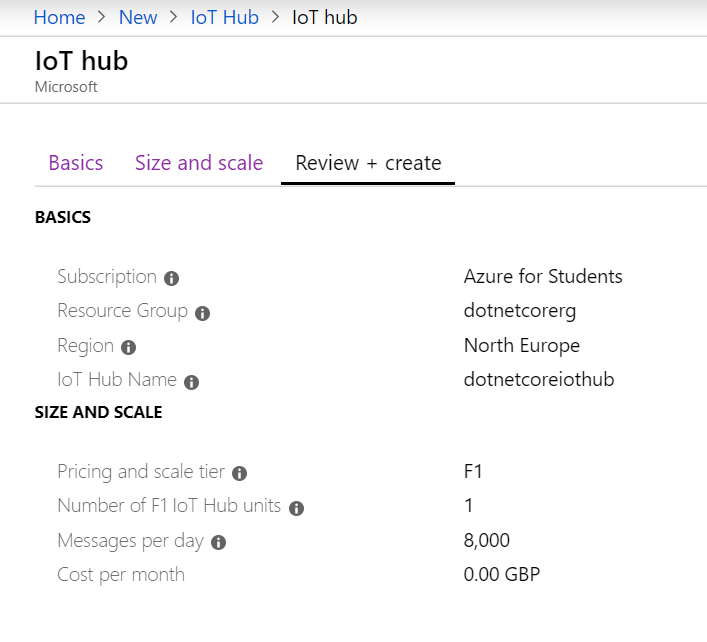


IoT Hub Size and Scale Settings

Here we can set a variety of options, however, we’re only really interested in the “Pricing and Scale Tier” setting. For our purposes, we’ll set this to “F1: Free Tier”. That way, there’ll be no charge at all.

* **Review and Create**

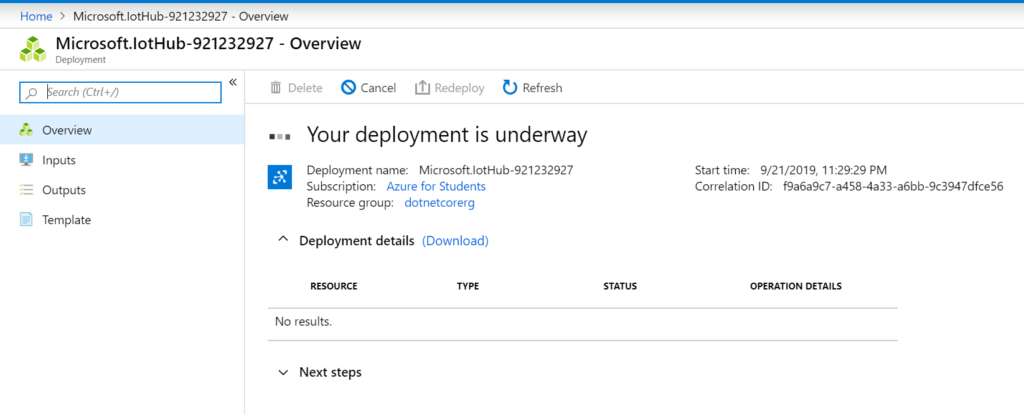
Press the “Review and Create” button, then check that all the settings are correct;



IoT Hub Review and Create

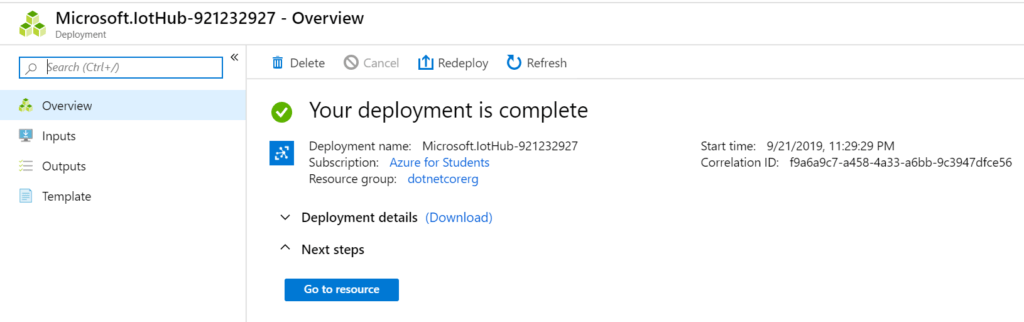
If you’re happy with the settings, then feel free to press the blue “Create” button at the bottom of the screen.

You’ll be taken to a holding page while the IoT Hub is created;



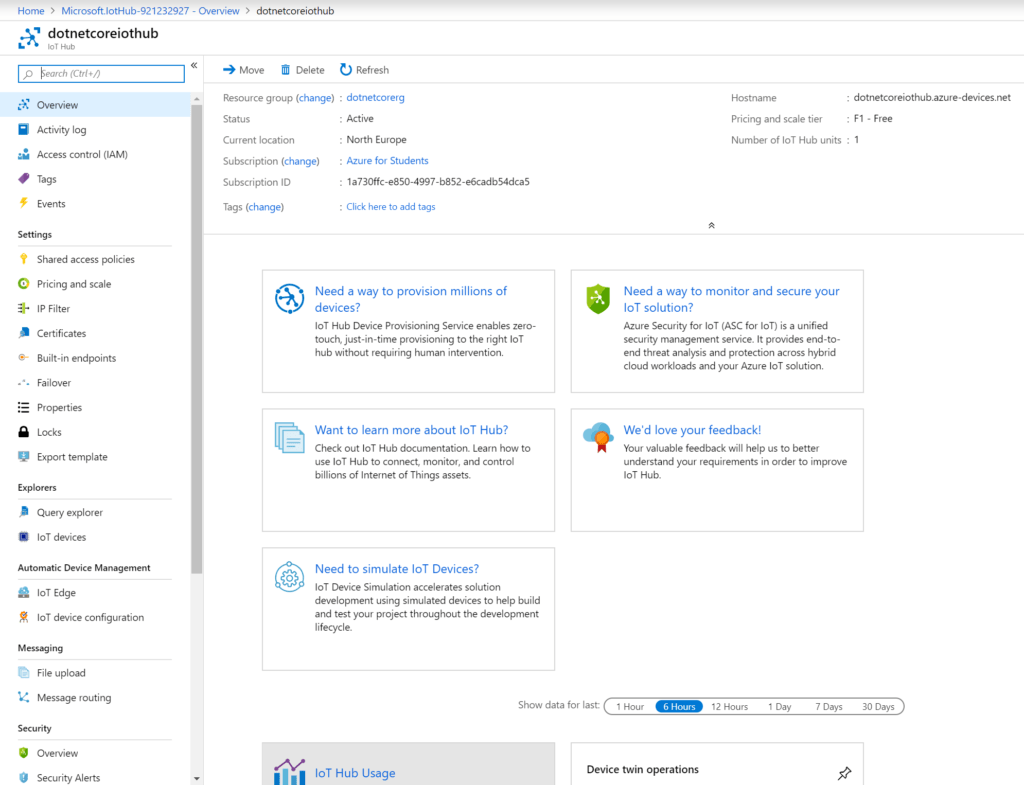
IoT Hub Creation

Once the IoT Hub is created, you’ll see the “Deployment Complete” screen;



IoT Hub Deployment Complete

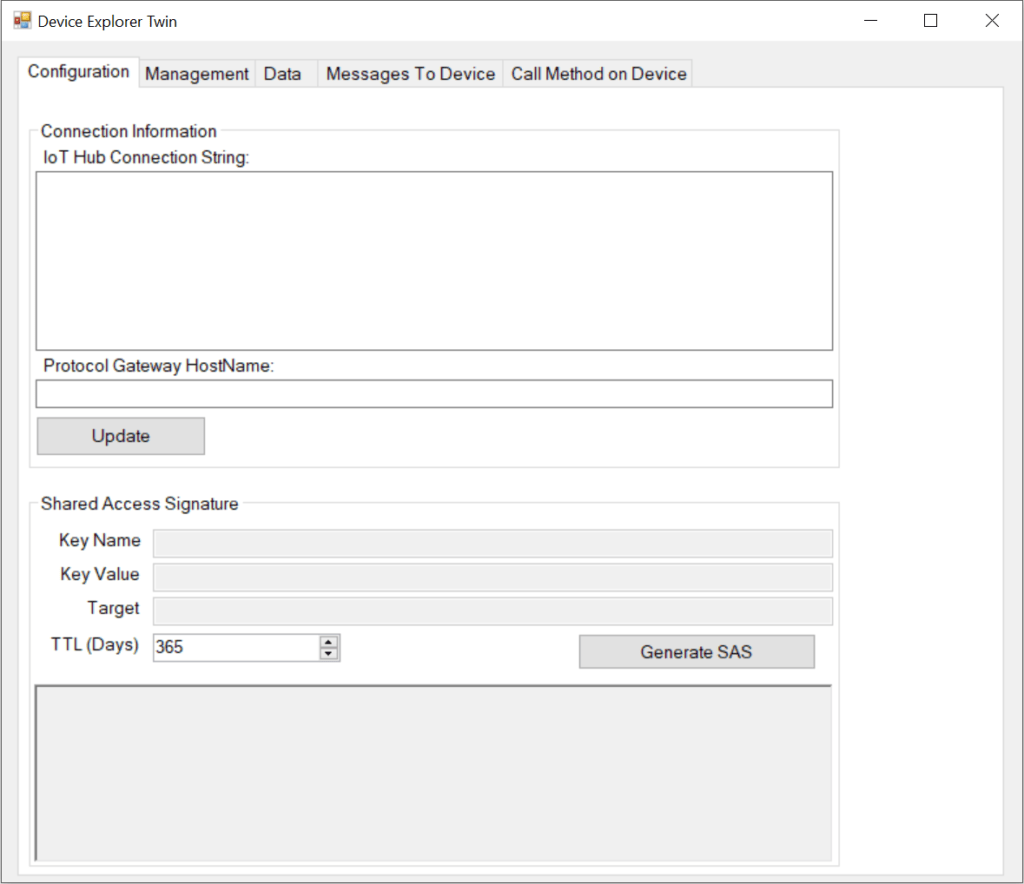
Hit the blue “Go to resource” button to navigate to our new IoT Hub;



IoT Hub Created

**IoT Hub SDK Device Explorer**

Now we have an IoT Hub setup, let’s get the IoT Hub Device Explorer running. If you haven’t done so already, and you’re running Windows, then [download the Windows Installer Package for the Device Explorer Application](https://github.com/Azure/azure-iot-sdk-csharp/releases/download/2019-1-4/SetupDeviceExplorer.msi). Launching the Device Explorer will show the Configuration Tab and a space for the IoT Hub Connection String;



IoT Hub Device Explorer

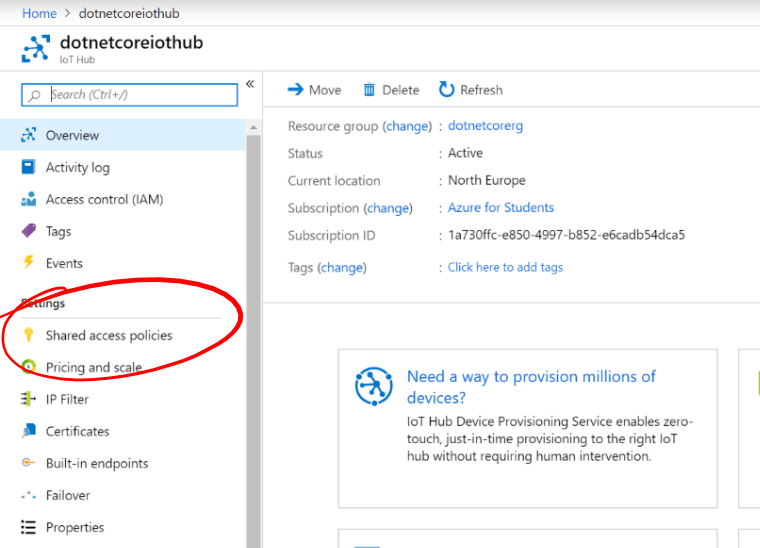
* **Device Explorer Connection String**

We’ll be using the Device Explorer to monitor messages between the IoT devices (Our Raspberry Pi) and the IoT Hub. We can also use it to manage individual devices, find connection strings, and also control individual IoT devices by sending messages and calling methods.

Because we need this much control, we’ll need a connection point which affords us overall control. For this we’ll use a Shared Access Policy which is an Azure IoT Hub level access point.

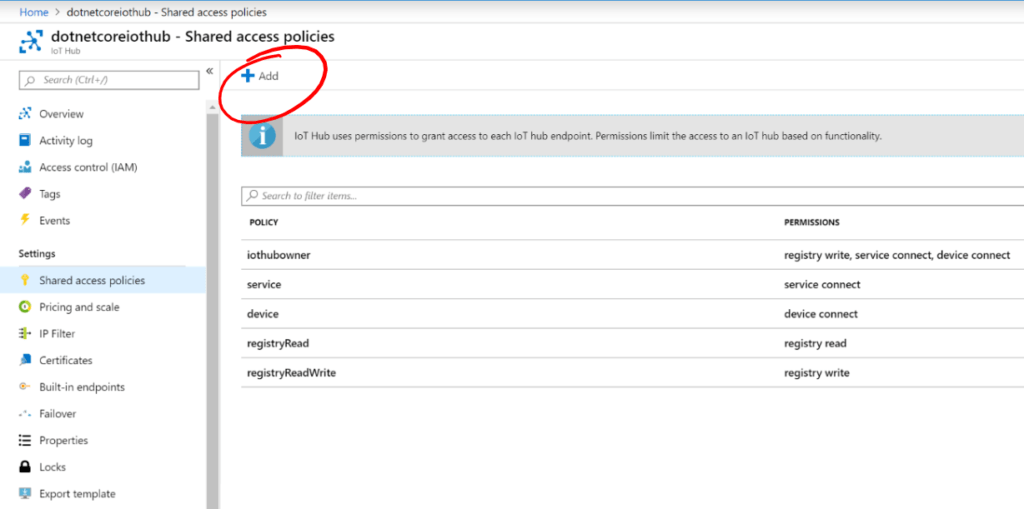
* **Shared Access Policy**

From the menu on the left of your IoT Hub, choose “Shared Access Policies”;



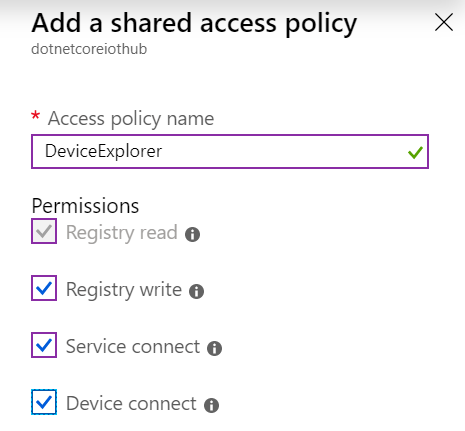
IoT Hub Shared Access Policies Menu Item

You’ll now see the IoT Hub Shared Access Policies screen. We’re going to create a new Access Policy for the IoT Device Explorer to connect to the IoT Hub. Hit the “+ Add” button in the top left;



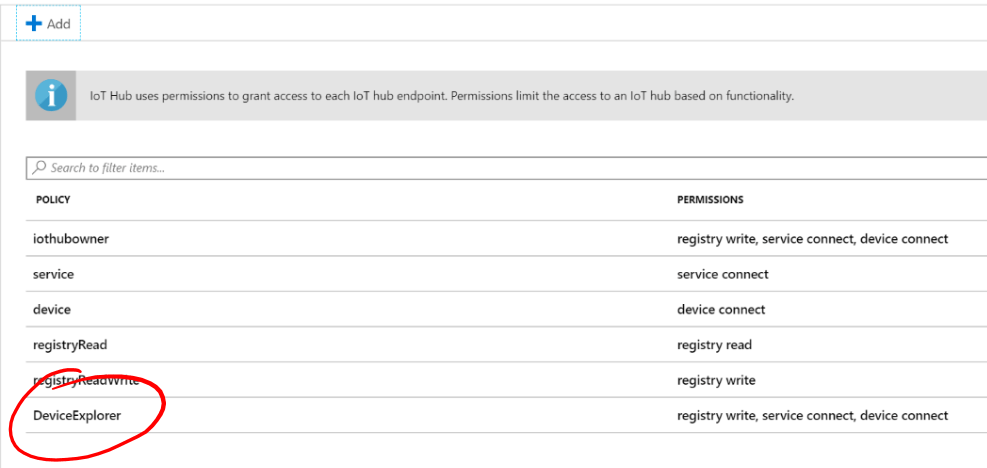
Add Shared Access Policy Button

We can now enter the details for our Shared Access Policy, where we can enter “DeviceExplorer” for the “”Access Policy Name”, select all of the checkboxes to give the Device Explorer full access and hit the blue “Create” button at the bottom right of the screen;



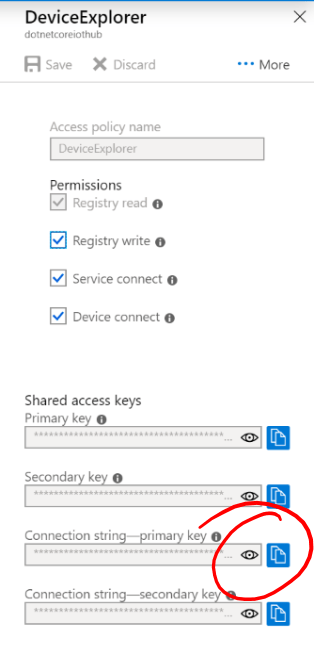
Device Explorer Shared Access Policy Settings

* **The new Shared access policy will be shown in the list;**



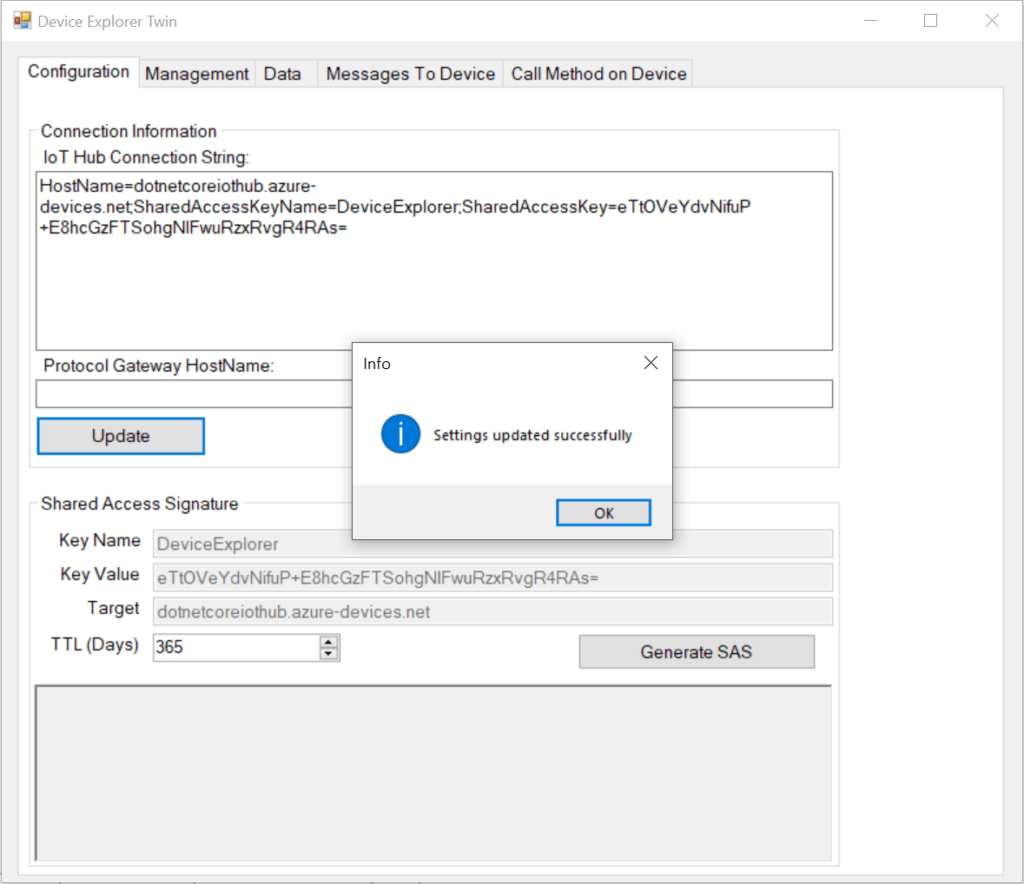
Shared Access Policy Lst

Click on the new DeviceExplorer Shared Access Policy, and hit the blue Copy button next to the “Connection String – primary key” box;



Device Explorer Shared Access Policy Details

Return to the Device Explorer and past the Connection String into the “Connection Information” box. Pressing the Update Button will , if all is well, show a message box saying “Settings updated successfully”;

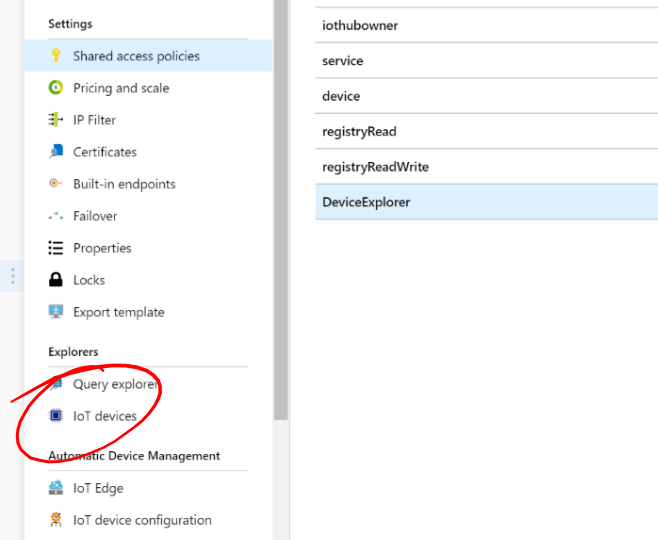


Device Explorer Settings Updated Successfully

**Add an IoT Device**

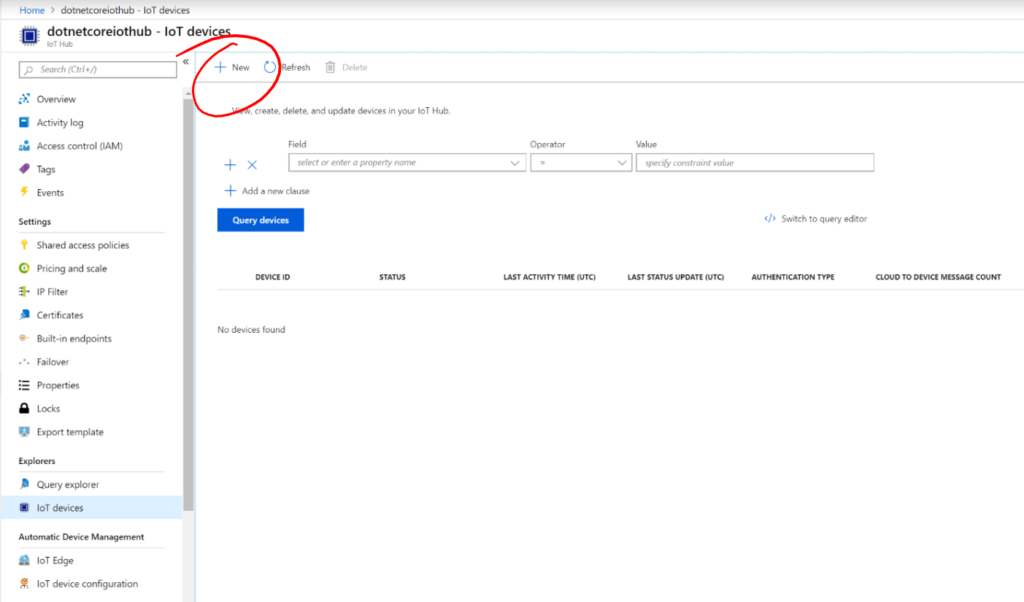
We now need to create an IoT Device for our Raspberry Pi to communicate with. Each physical IoT device (in our case a Raspberry Pi), get’s it’s own corresponding IoT Hub Device to communicate with.

To create an Azure IoT Device, return now to the Azure Portal and to your IoT Hub and click the “IoT Devices” menu item on the left;



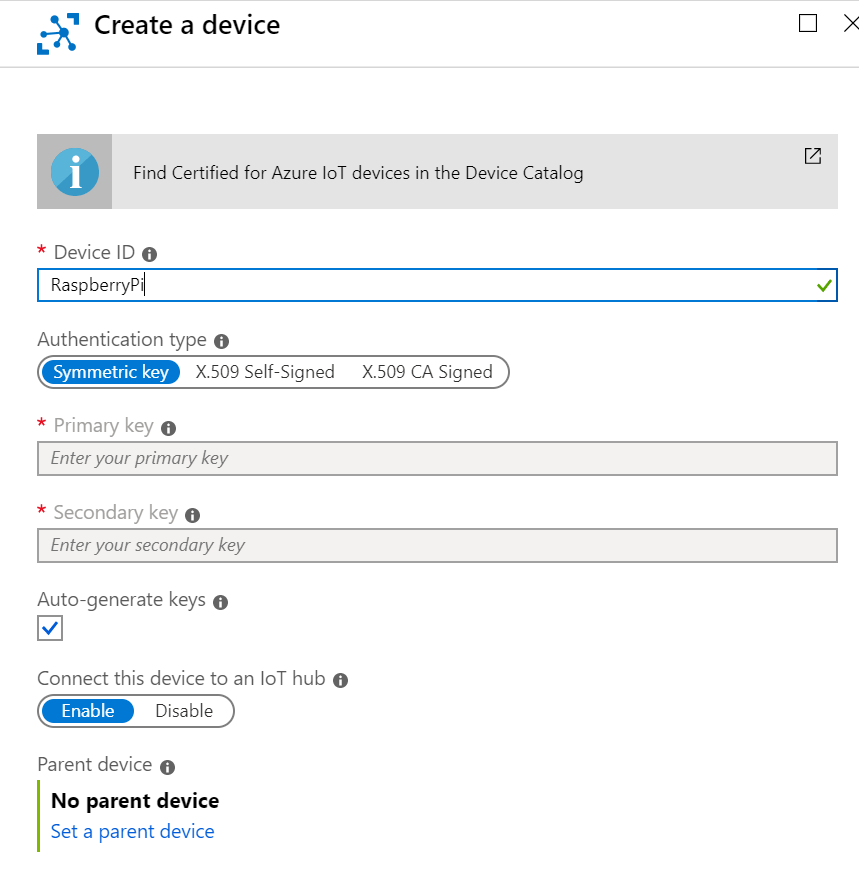
IoT Devices Menu Item

You’ll now be shown the IoT Devices List. Create a new IoT Device by hitting the “+ New” button at the top left;



IoT Hub IoT Devices

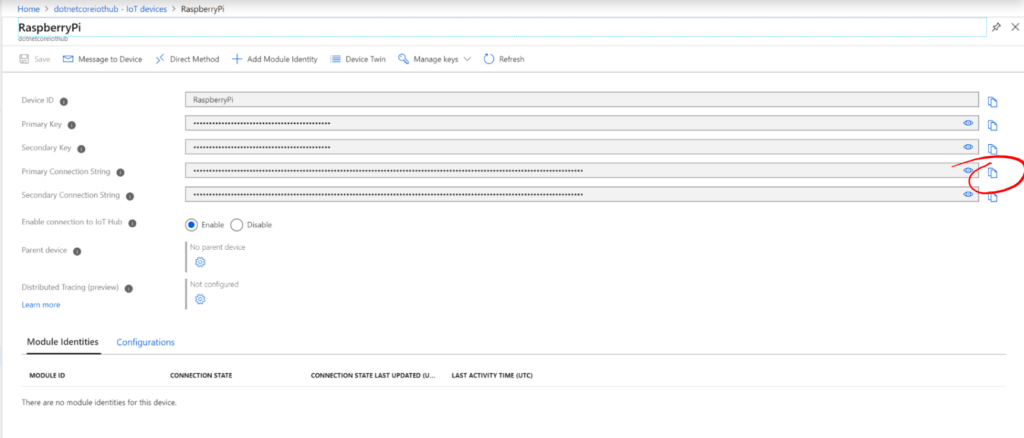
You’ll then see the “Create a Device” screen. Enter a name for your device, for instance “RaspberryPi”, leave the other settings as they are and hit the blue “Save” button in the bottom left;



Create a Device Screen

You’ll now see that the new “RaspberryPi” device will be shown in the, if your device isn’t shown, then hit the “Refresh” button. Now click on our new “RaspberrryPi” device to access it’s properties;

We can now copy the new IoT Device’s Connection String by hitting the blue Copy button next to the “Primary Connection String” box;

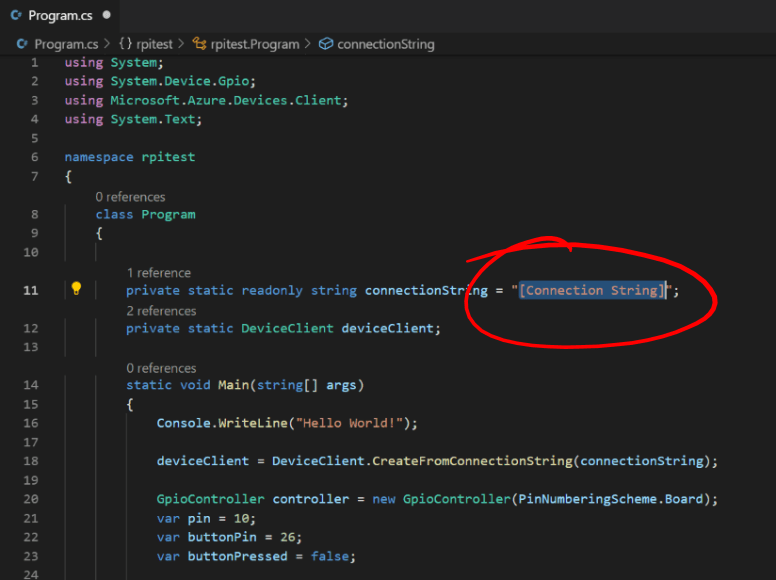


IoT Device Properties

## **Filling in the Blanks**

We now need to fill in the connection string for our Raspberry Pi. Return to VS Code and our “program.cs”.

Replace the [Connection String] text with your copied IoT Device Connection String (make sure to keep the quotes!”;



Replace Connection String

Save your file and return to your PuTTY terminal window.

## **Run Time!**

Make sure you’re in the “share/rpitest” directory, make sure your code builds by running;

dotnet build

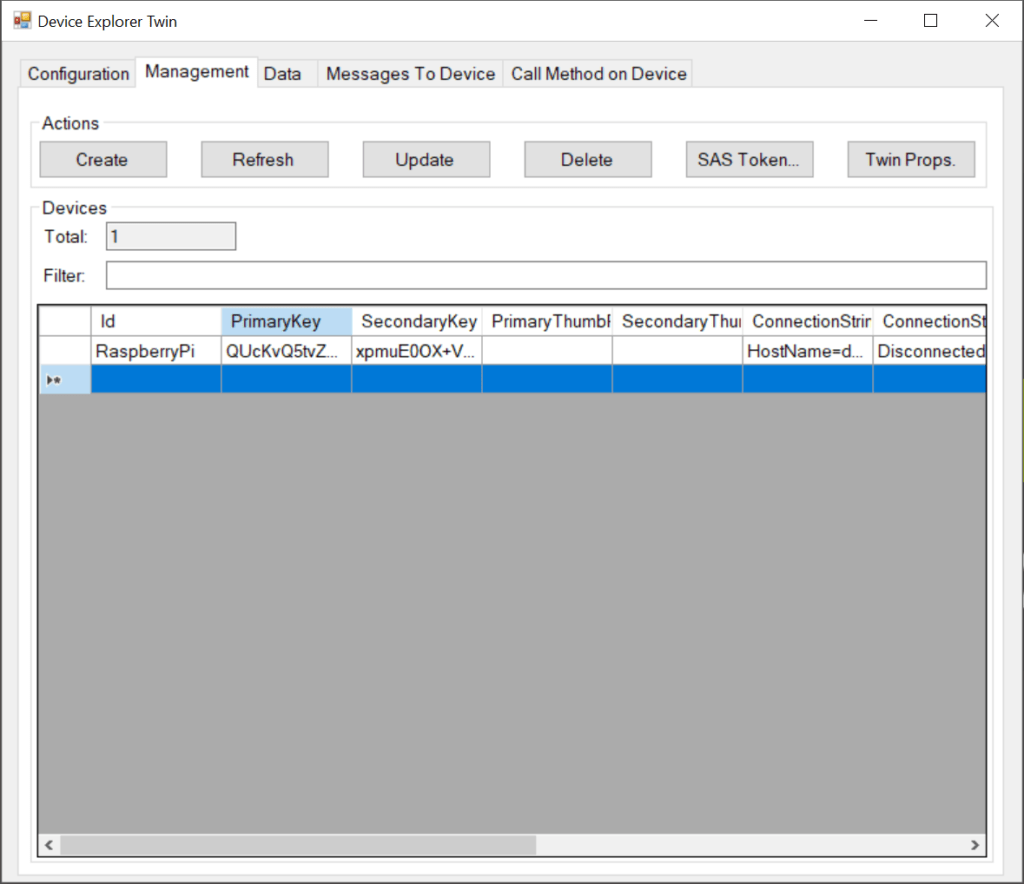
All being well, the code will build successfully;

If your code builds ok, then run the following to run the new code;

dotnet run

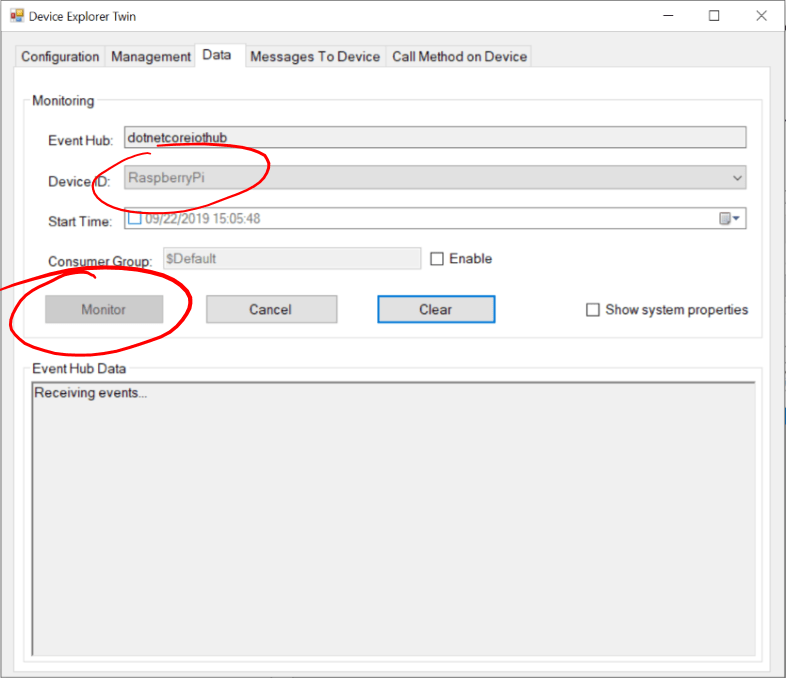
Once your code is running, get both the Device Explorer and Terminal Window on the screen at the same time.

If you click the “Management” tab in the Device Explorer then you’ll see your RaspberryPi IoT Hub Device;



Device Explorer Management Tab

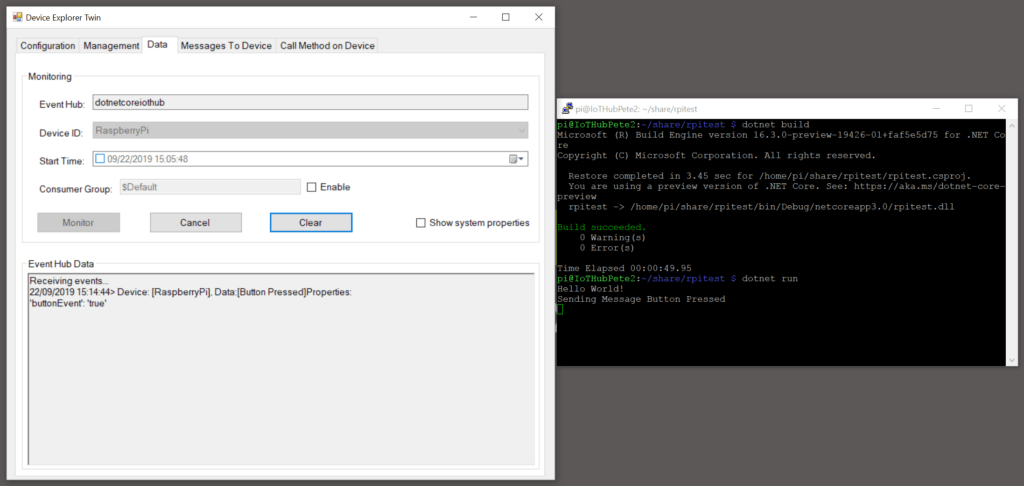
Now click the “Data” tab and make sure that your “RaspberryPi” device is shown in the “Device Id” dropdown. Then hit the “Monitor” button to start monitoring IoT Hub messages. The “Event Hub Data” box should show “Receiving events…”;



Device Explorer Data Tab

## **Push the Button**

Now, push the button on the board and you should see a message in the Terminal Window and also a message received by the Device Explorer;

IoT Hub Messages