## References

* Microsoft Azure

[https://azure.microsoft.com](https://azure.microsoft.com/)

* Azure Portal

[https://portal.azure.com](https://portal.azure.com/)

* Microsoft Azure IoT SDKs

<https://github.com/Azure/azure-iot-sdks>

* How to process IoT Hub device-to-cloud messages using .Net

<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-csharp-csharp-process-d2c>

## Requirements

* Azure account with subscription
  + Free trial: <https://azure.microsoft.com/en-us/free/>
* Download and install Device Explorer tool (Windows GUI)

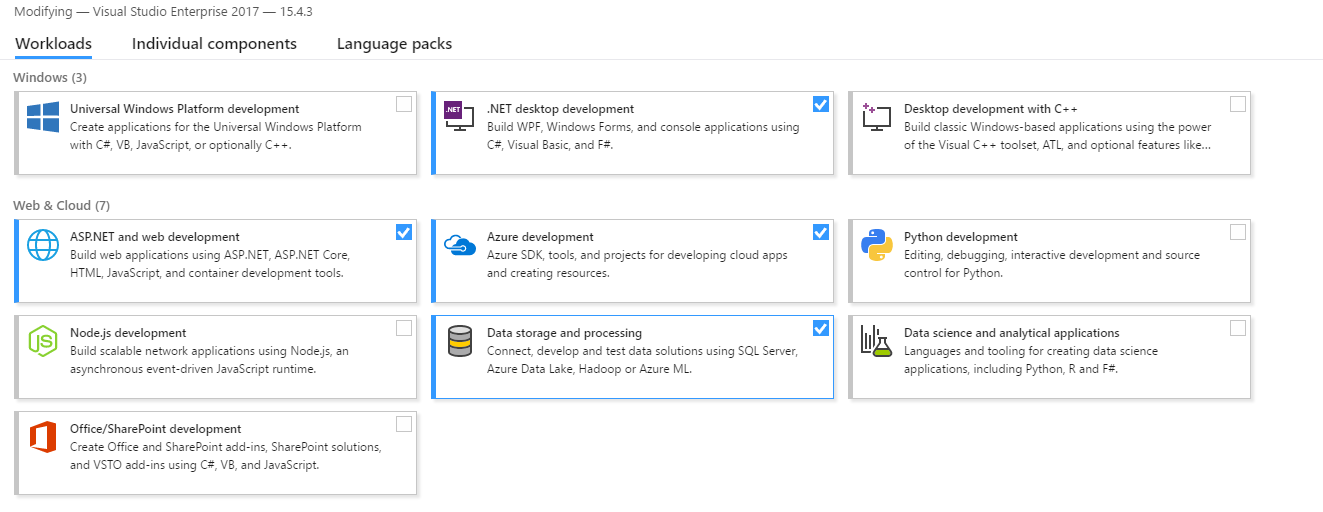
Open source:

<https://github.com/Azure/azure-iot-sdk-csharp/tree/master/tools/DeviceExplorer>

Prebuilt file:

<https://github.com/Azure/azure-iot-sdk-csharp/releases/download/2017-10-6/SetupDeviceExplorer.msi>

* Microsoft Visual Studio 2017 with workloads
  + .NET desktop development
  + ASP.NET and web development
  + Azure Development
  + Data storage and processing

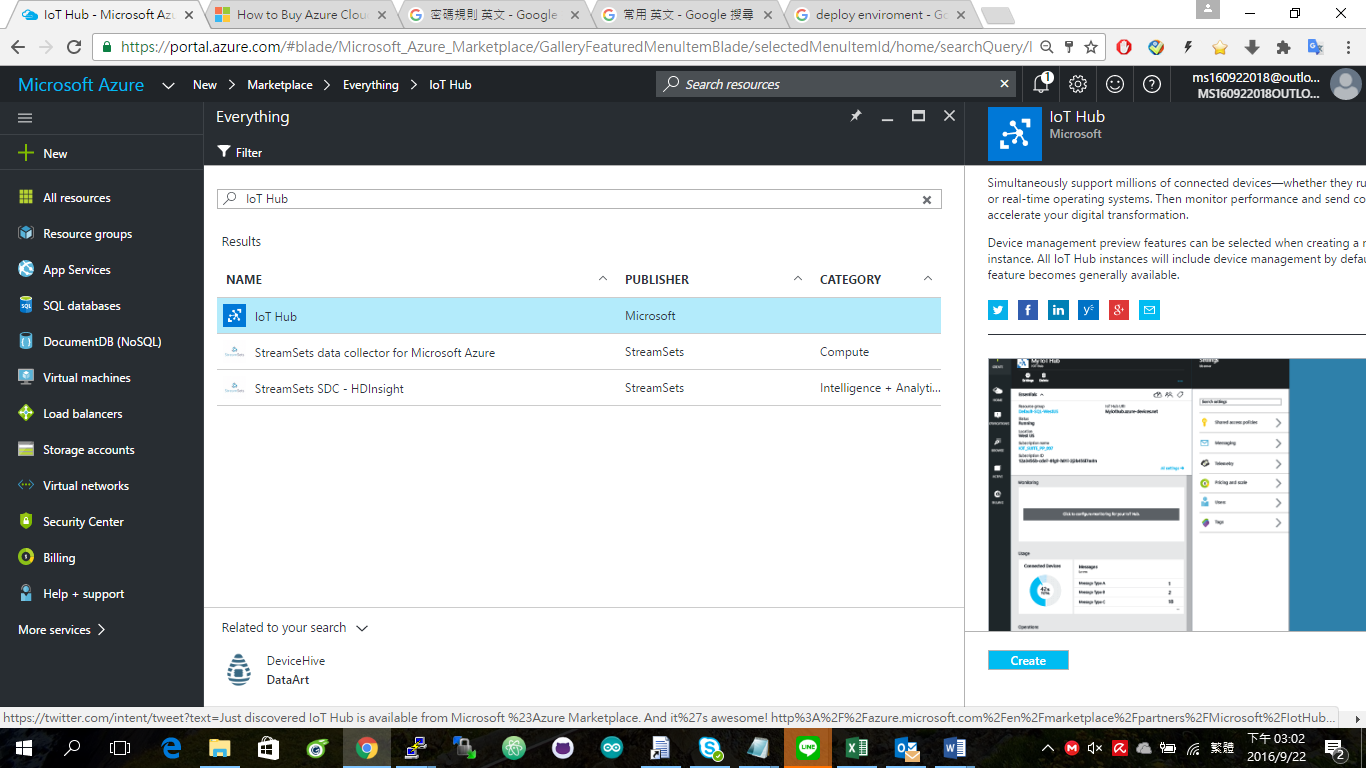


## Goals

* Provision an Azure IoT Hub
* Build the 2 simulated devices with C# and Node.js SDKs
* Observe the data communication between Device-to-Cloud (D2C) and Cloud-to-Device (C2D) via Device Explorer tool

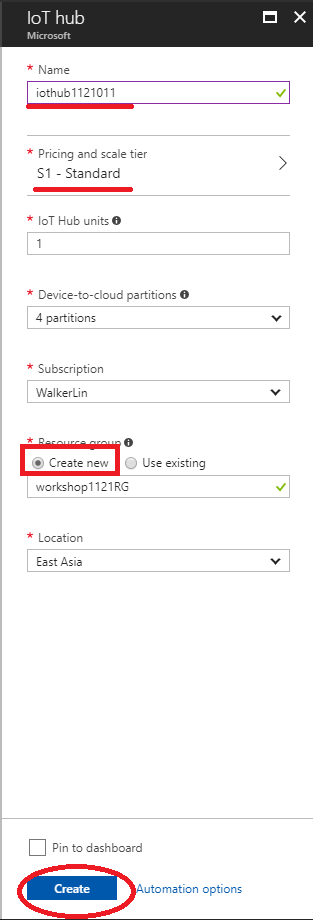
## Step 1: Provision an Azure IoT Hub

* Login <https://portal.azure.com>
* Search **IoT Hub** and create new one.



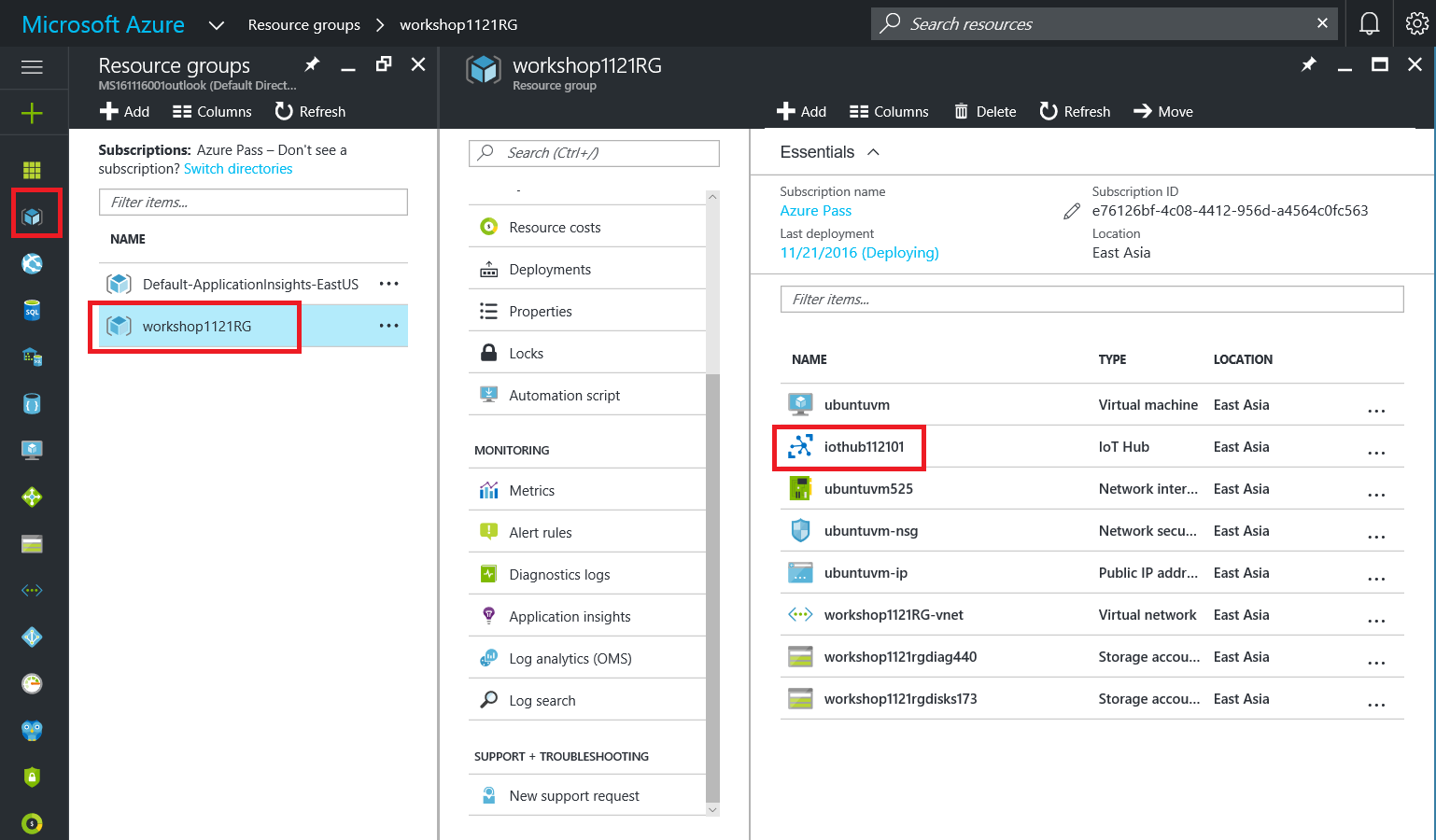
Here we select **S1 Standard** for this workshop

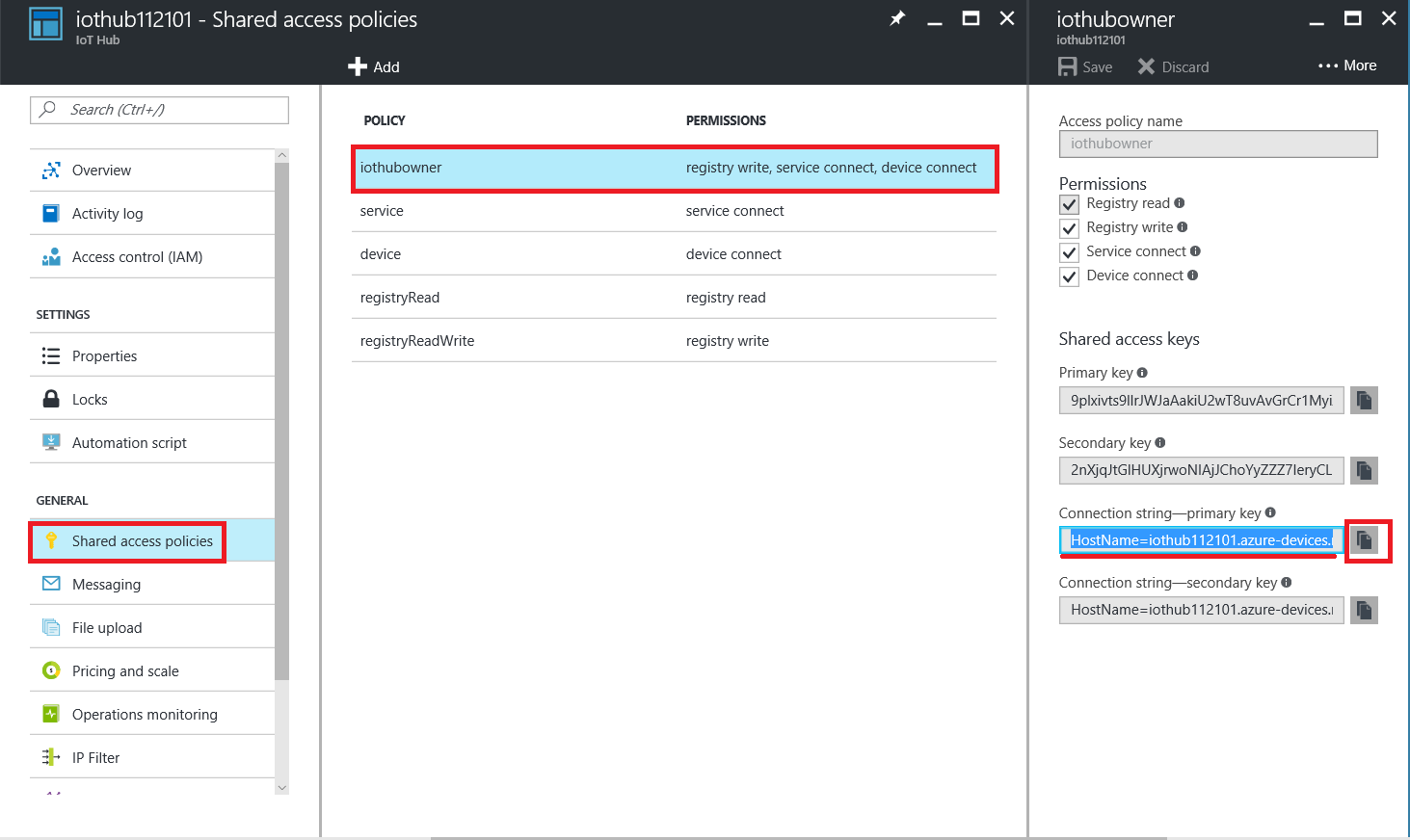
* + **Create** a new resource group to manage your solution.
  + Select a location (service region), for example, **East Asia**.



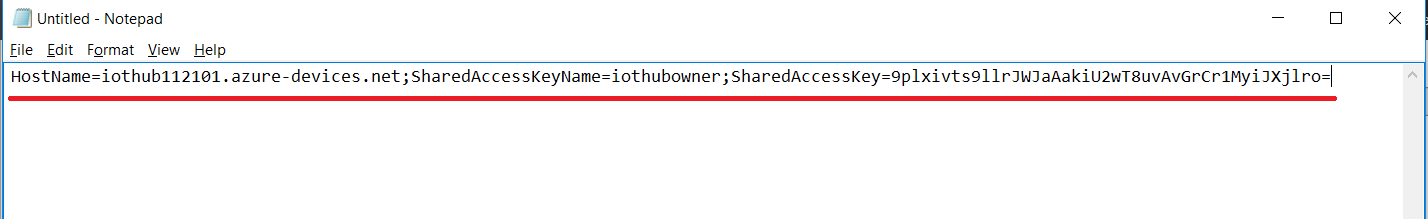
(It may take a few minutes to deploy IoT Hub)

* Next, we need to get the connection string of IoT Hub
  + Navigate to your IoT Hub.
  + Select the **Share access policies**, then choose the policy **iothubowner**.
  + Copy the **Connection String - primary key**



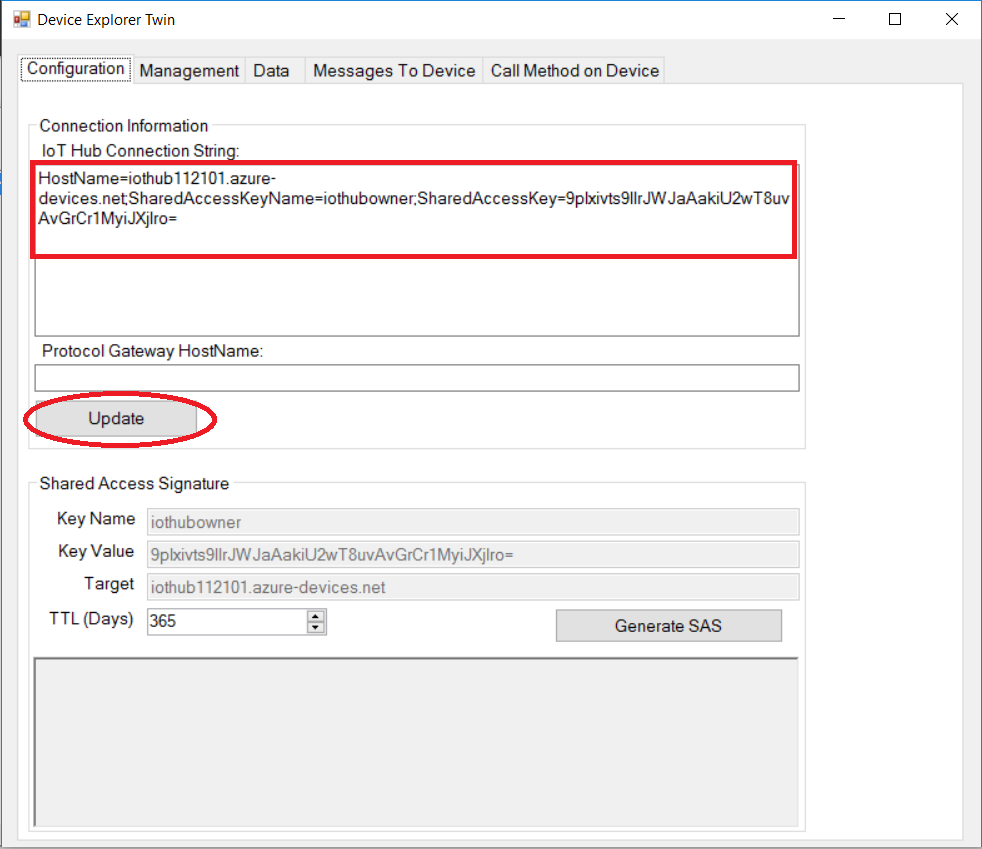


* + Save the **Connection String - primary key** for the later used.

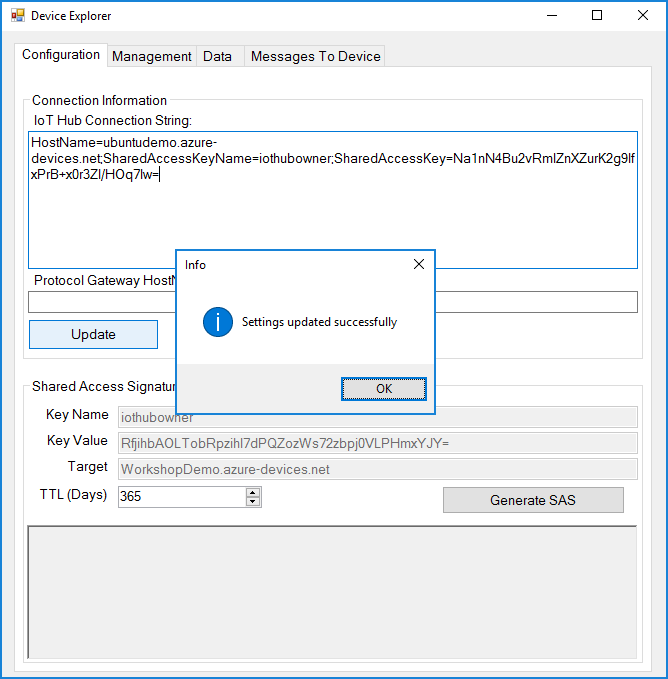


## Step 2: Provision 2 devices on IoT Hub

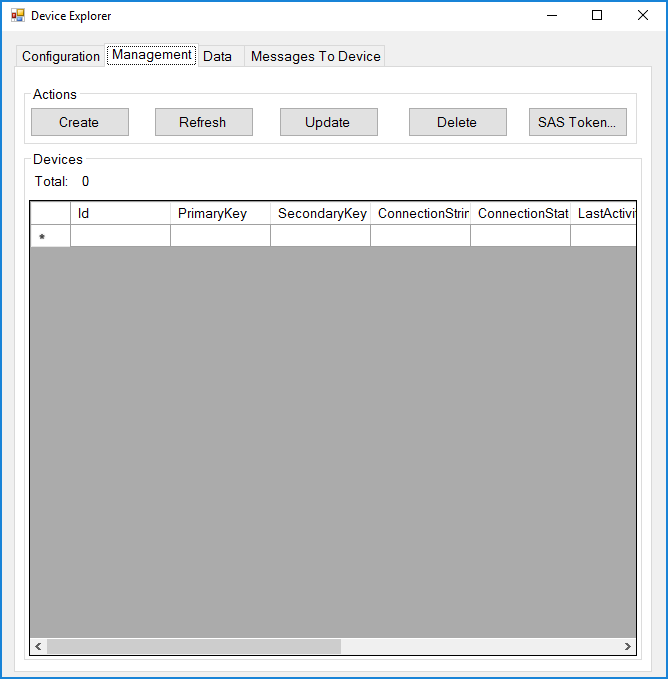
* Using Device Explorer Tool to provision your device in this IoT hub and get the connection string of device.
* Open this tool, and find the first tab **Configuration**, paste the Connection string of IoT Hub then click **Update**.



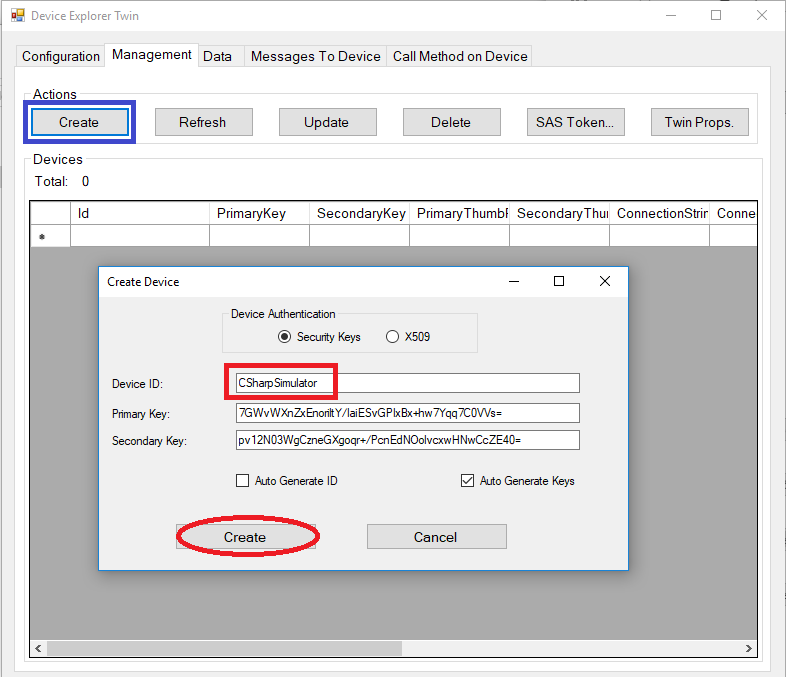
* + Update successfully



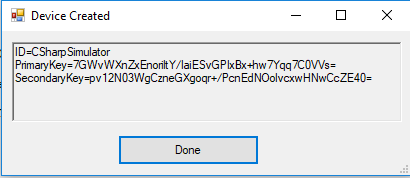
* In the second tab **Management**, here you can manage all devices which are connected to this IoT Hub.



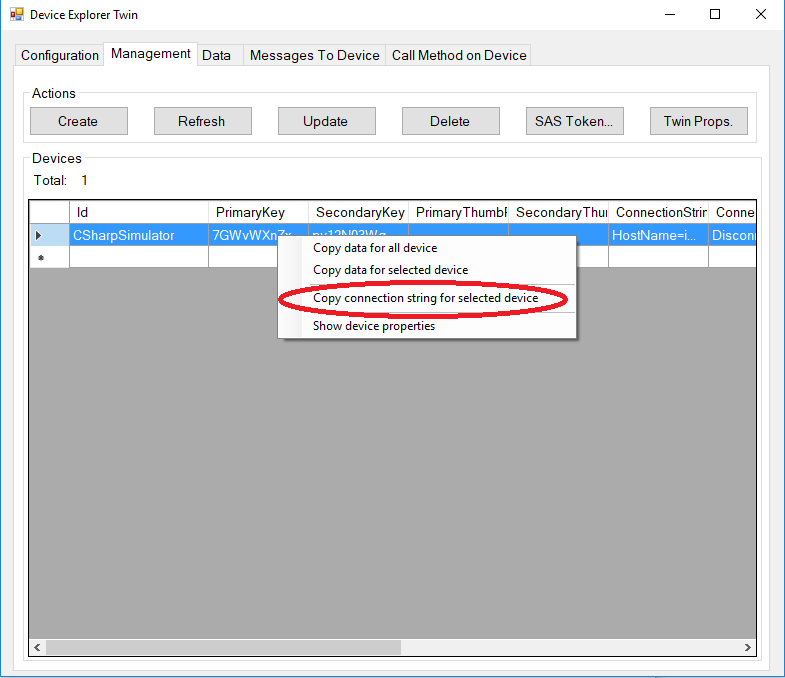
* **Create** the first simulated device. Named it to **CSharpSimulator**. (depends on your favorite name)



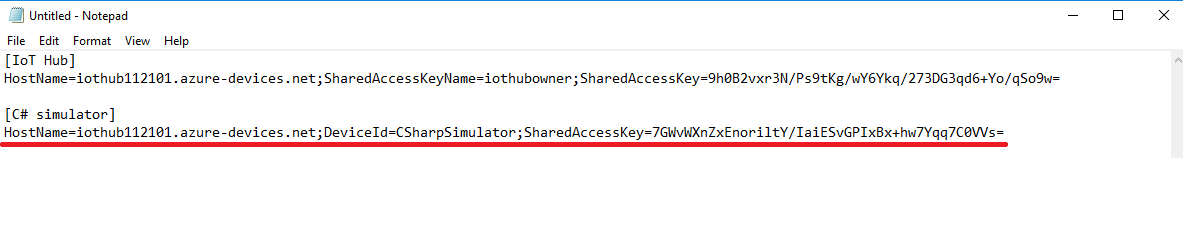
* + Created successfully



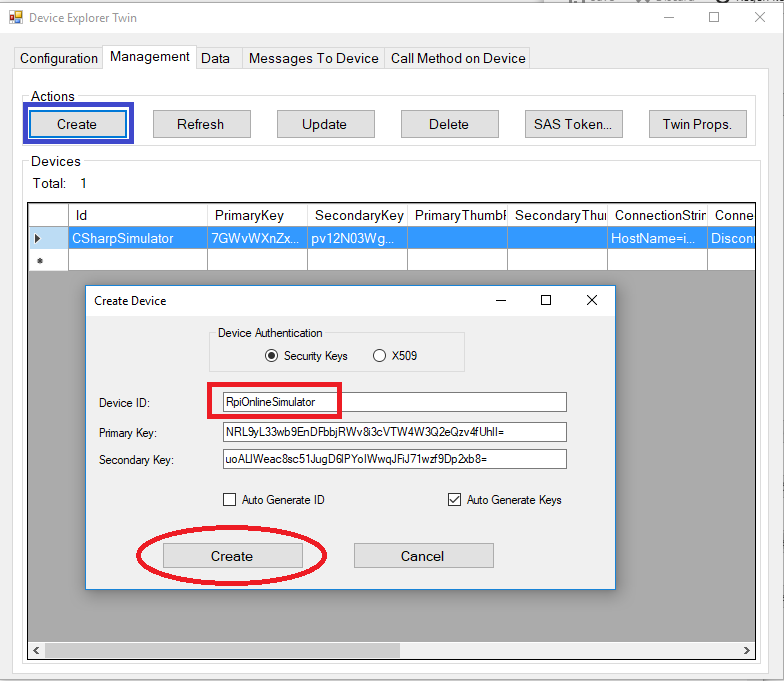
* When the list was updated, we **select** the **CSharpSimulator** and right-click to copy the connection string of device.

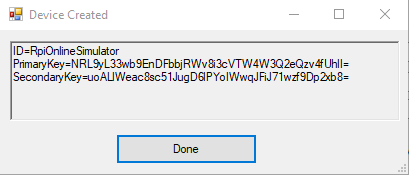


* + Take a note with this connection string of device, and it will be used later.

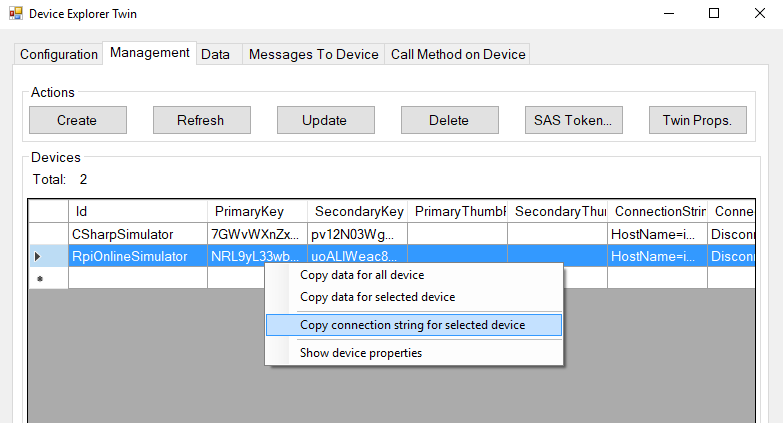


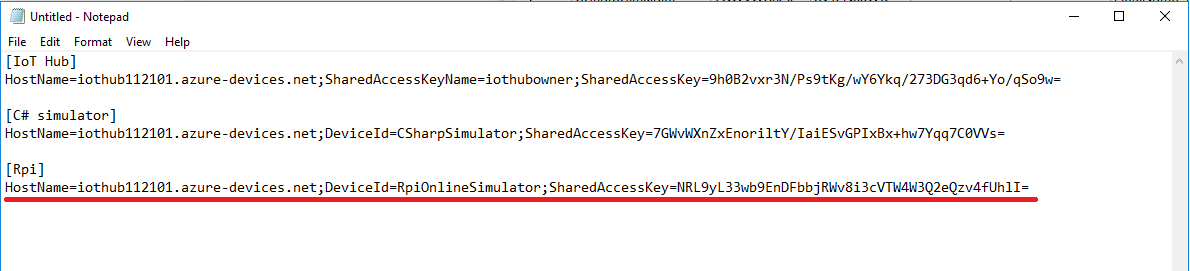
* Create the second device for **RpiOnlineSimulator.**





* Copy the connection string of device and save it for later used.

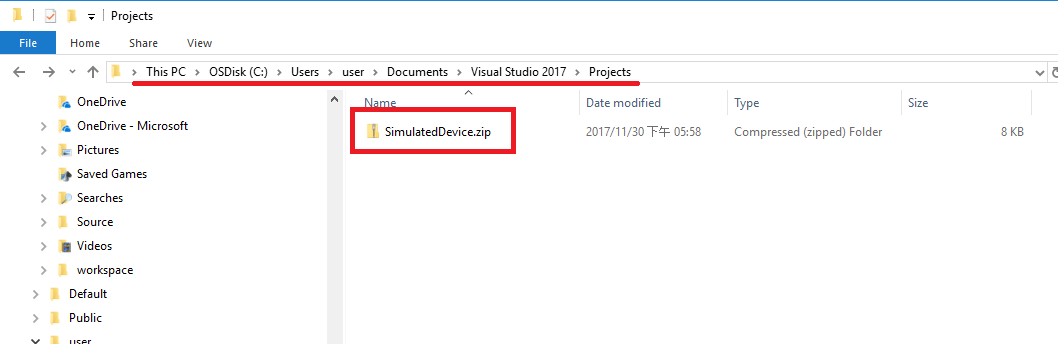




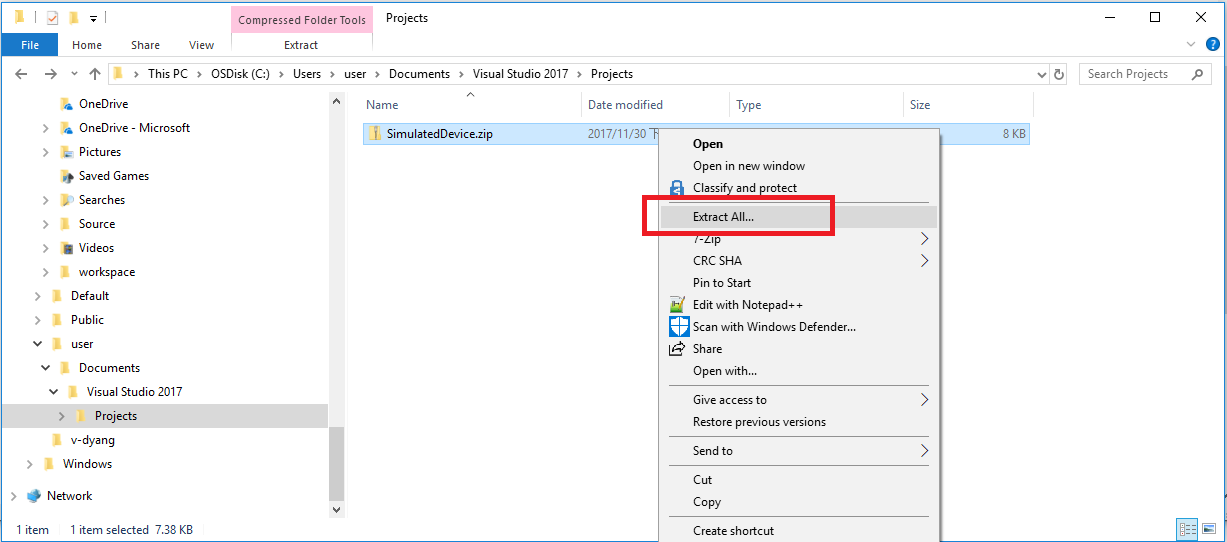
## Step 3: Build the C# simulated device

* Copy and extract the simulated device of C# project (**SimulatedDevice.zip**) to the Projects of Visual Studio (VS).

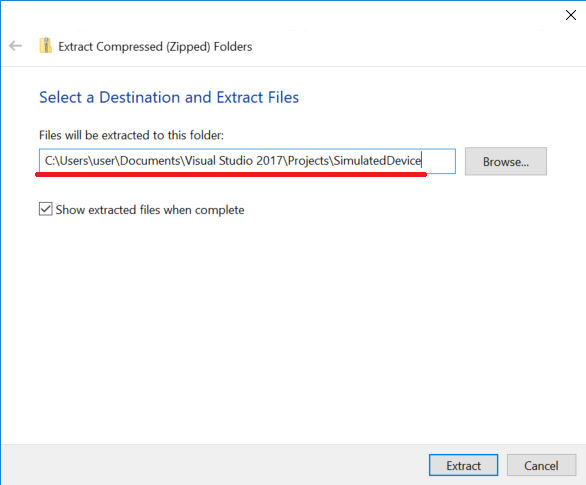
Path: **C:\Users\<username>\Documents\Visual Studio 2017\Projects\**



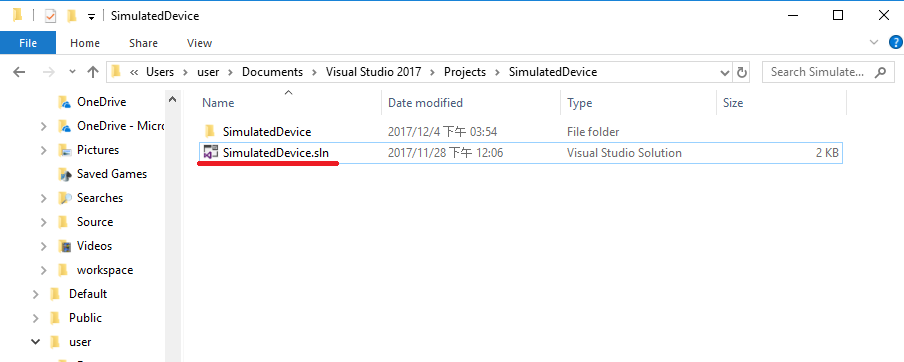
* Unzip



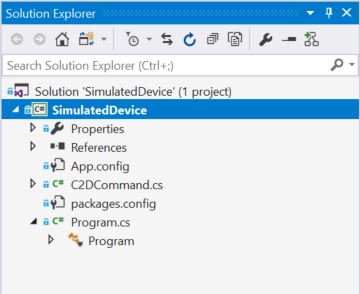
* The files should be extracted to **C:\Users\<username>\Documents\Visual Studio 2017\Projects\**

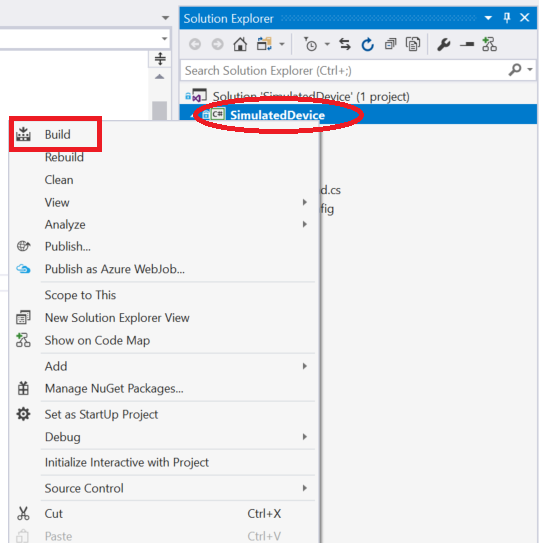


* Open the **SimulatedDevice** solution (.sln file) of VS

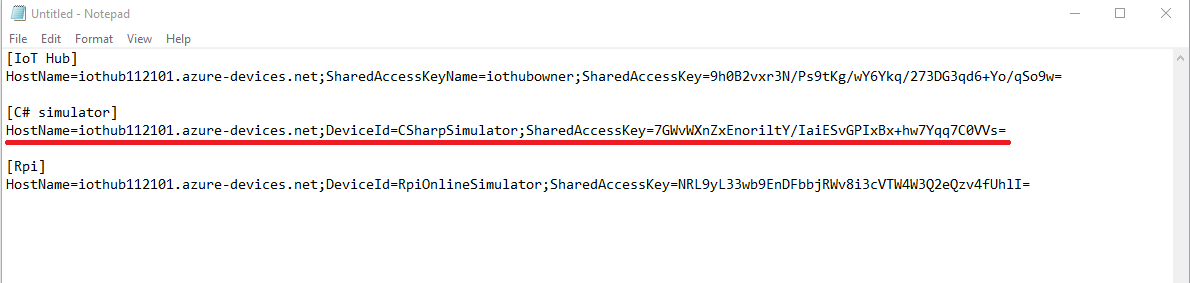


* Build the C# simulator
  + Right-click to build the project to restore all NuGet packages.

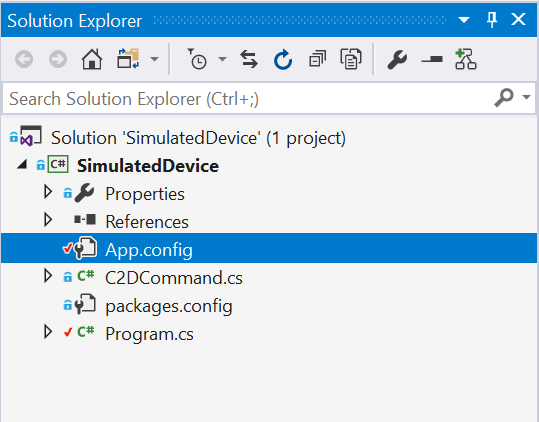


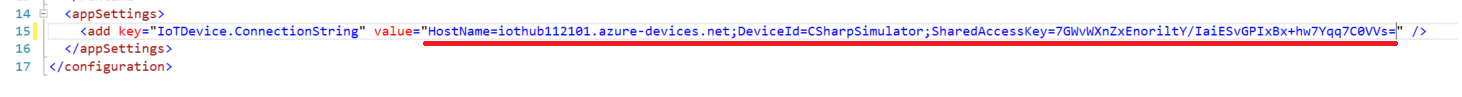


* + Get the connection string of CSharpSimulator



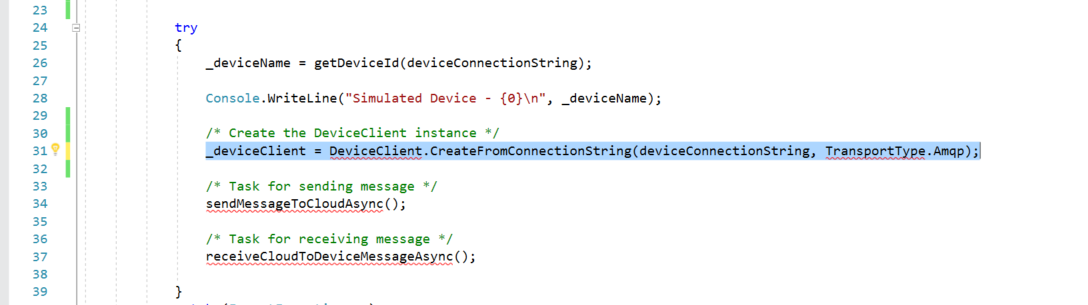
* + Paste the device connection string into the value of IoTDevice.ConnectionString in **App.config** file.



****

* + Open the **Program.cs**, then create a DeviceClient instance from Azure IoT Hub C# SDK.

\_deviceClient = DeviceClient.CreateFromConnectionString(deviceConnectionString, TransportType.Amqp);



* + Use **SendEventAsync** method to send message to IoT Hub.

await \_deviceClient.SendEventAsync(message);



* + Use **ReceiveAsync** method to receive Cloud-to-Device message from IoT Hub.

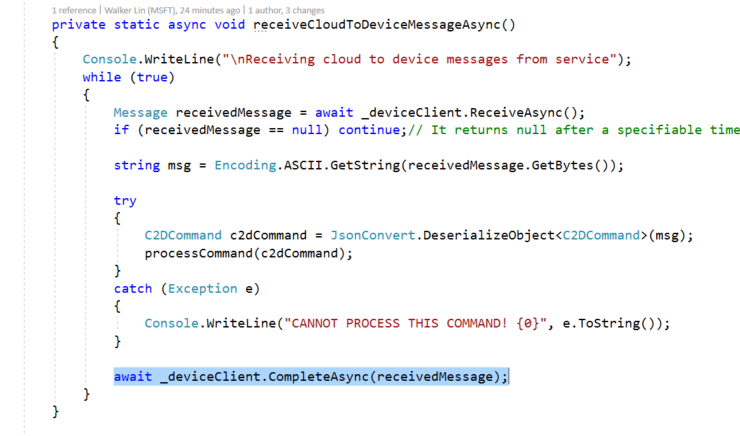
Message receivedMessage = await \_deviceClient.ReceiveAsync();

if (receivedMessage == null) continue;// It returns null after a specifiable timeout period (in this case, the default of one minute is used)

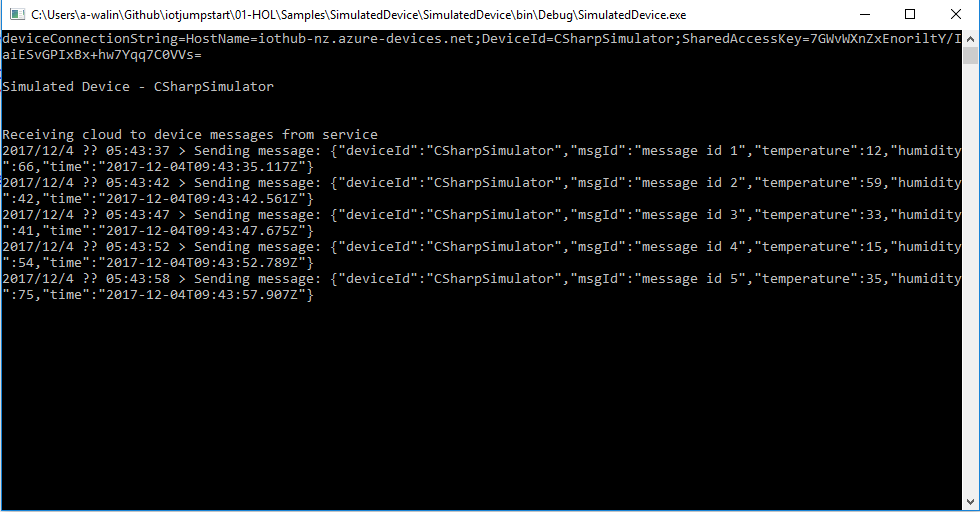


* + Complete the C2D message to IoT Hub.

await \_deviceClient.CompleteAsync(receivedMessage);



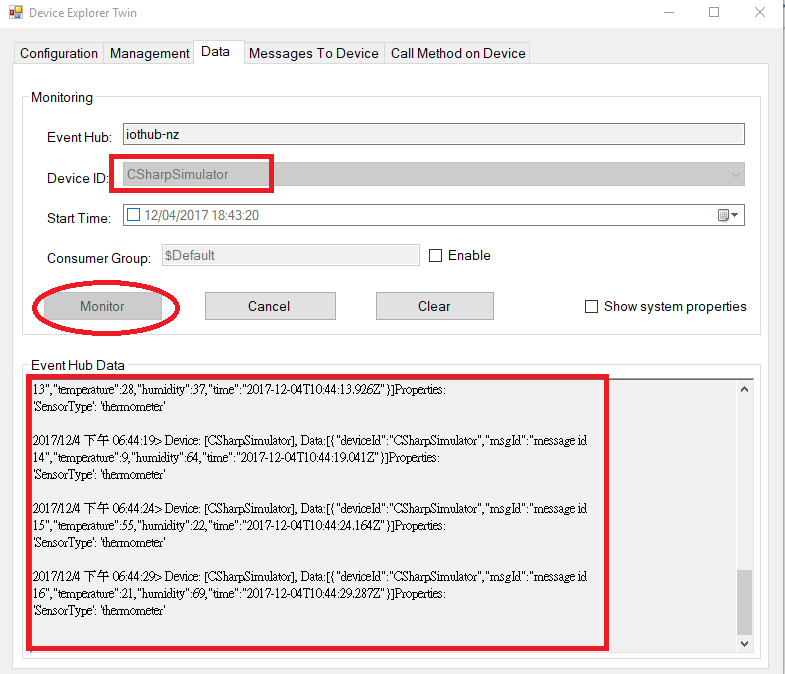
* Let’s press the **F5** to start debugging and check the output of console
  + Output



(Send the telemetry data every 5 seconds)

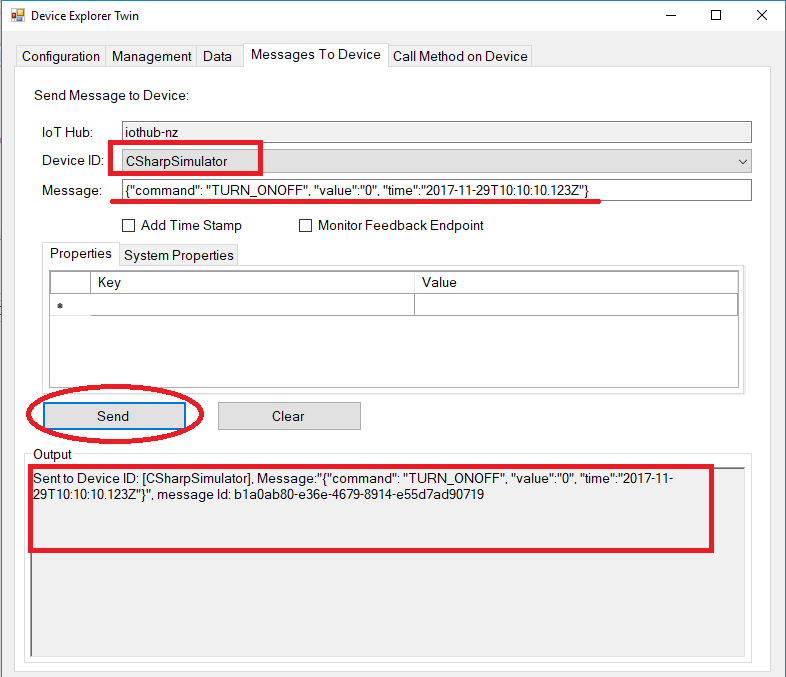
## Step 4: Observe the data communication between device and cloud

* Switch to **Data** tab, and select the device then observe the receiving events



**Note: The data fields include the deviceId, msgId, temperature, humidity and time. We also set a property for sensorType as thermometer for the later data processing.**

* Switch to **Messages To Device** tab, then test the Cloud-to-Device (C2D) messages.



You can use the following commands (JSON string) for testing.

(see the **c2d-command.txt** under **Scripts** folder)

{"command": "TURN\_ONOFF", "value":"1", "time":"2017-11-29T10:10:10.123Z"}

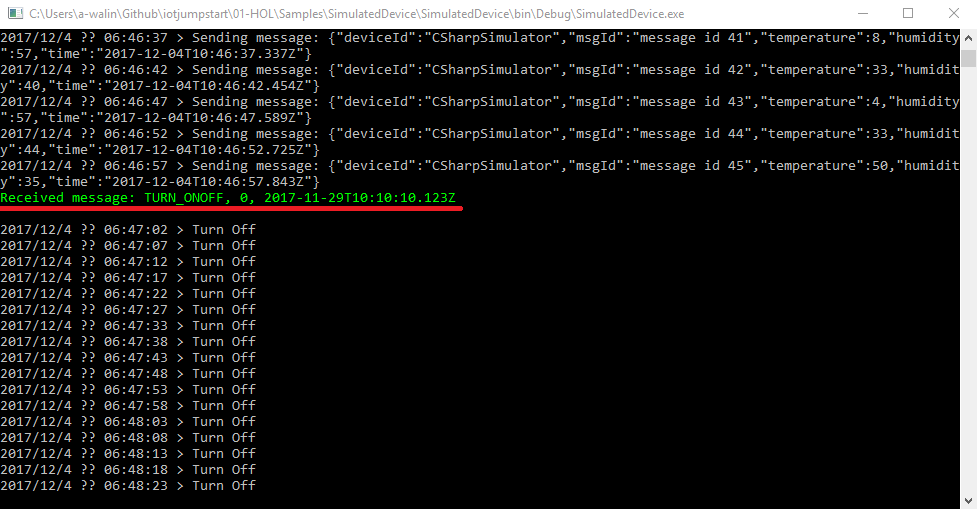
or

{"command": "TURN\_ONOFF", "value":"0", "time":"2017-11-29T10:10:10.123Z"}

or

{"command": "TEMPERATURE\_ALERT", "value":"50", "time":"2017-11-29T10:10:10.123Z"}

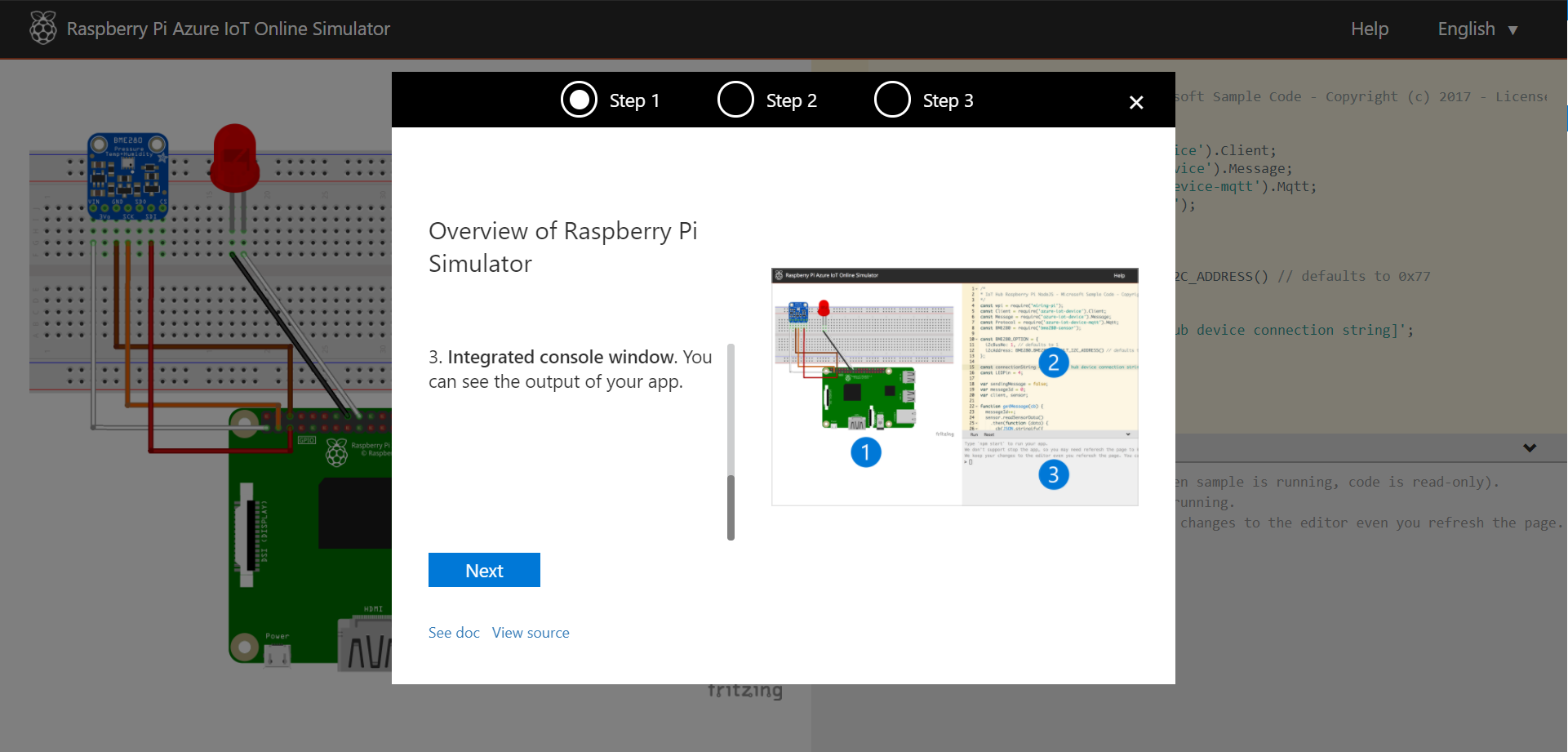
* The output of Windows Console App



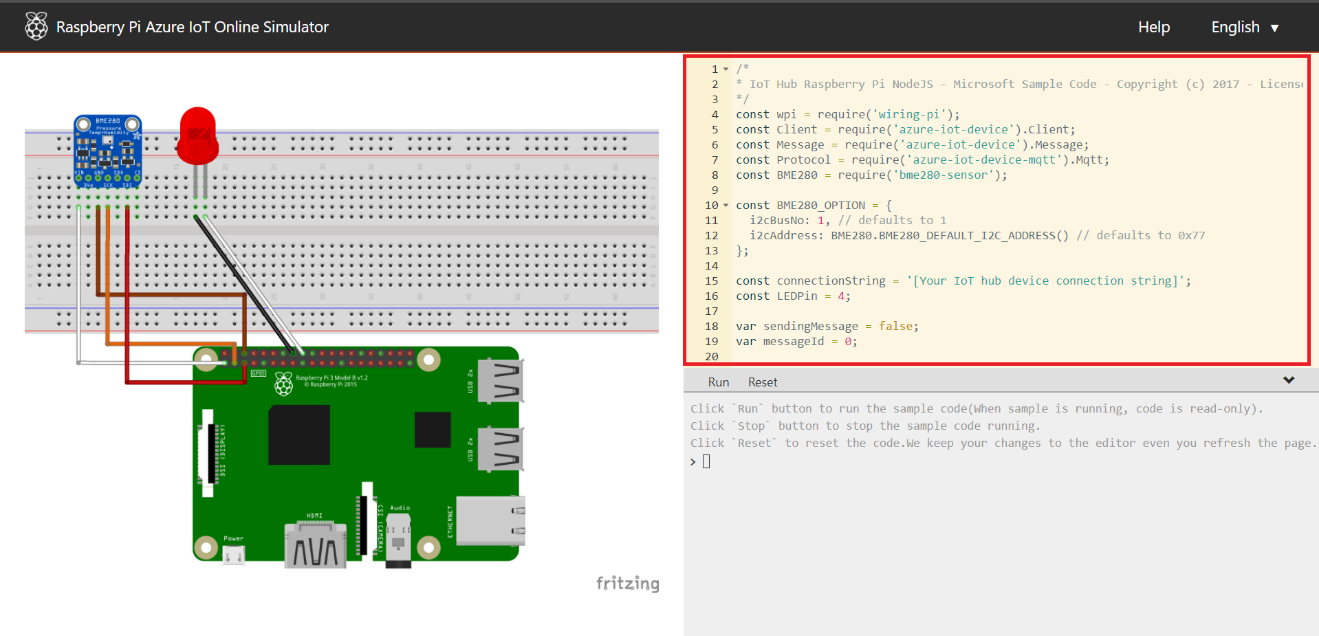
## Step 5: Run the second simulator on Raspberry Pi Azure Online Simulator

* Open this website on your browser

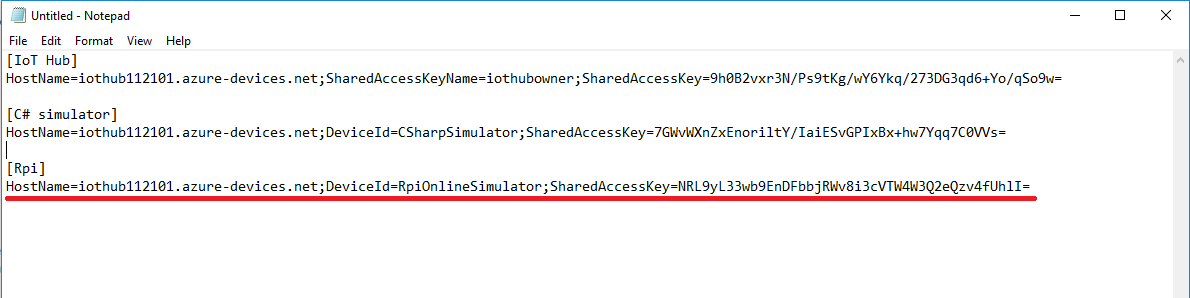
<https://azure-samples.github.io/raspberry-pi-web-simulator/>

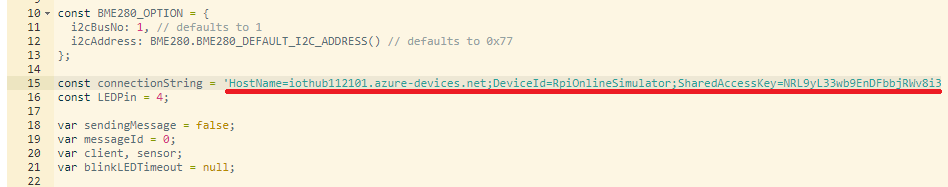


* Finish the tutorial, then you can start to write your Node.js code here.

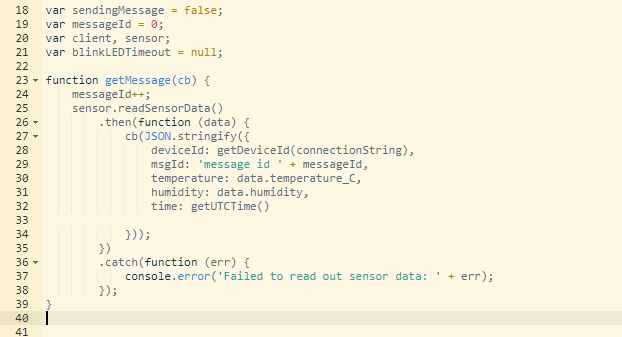


* Coding now!
  + Open the **RPi\_Simulator.js** under 01-HOL/Samples/Raspberry Pi Azure IoT Online Simulator, then copy all and paste it into the code area.
  + Put your device connection string in line 15.

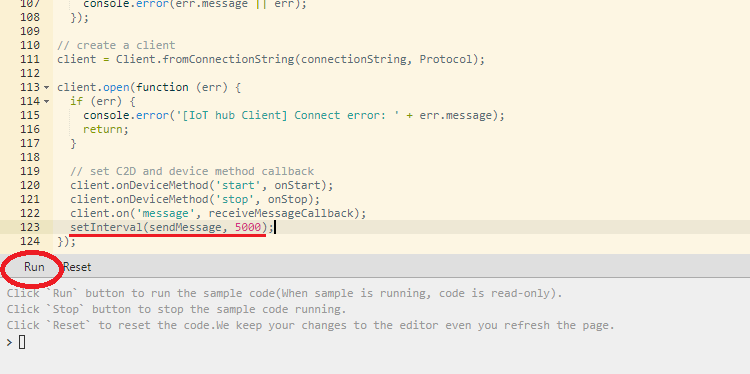




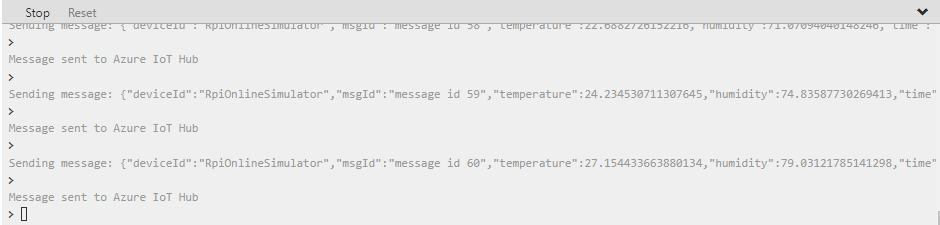
* + Check your telemetry data format on getMessage method.
    - **deviceId**
    - **msgId**
    - **temperature**
    - **humidity**
    - **time**



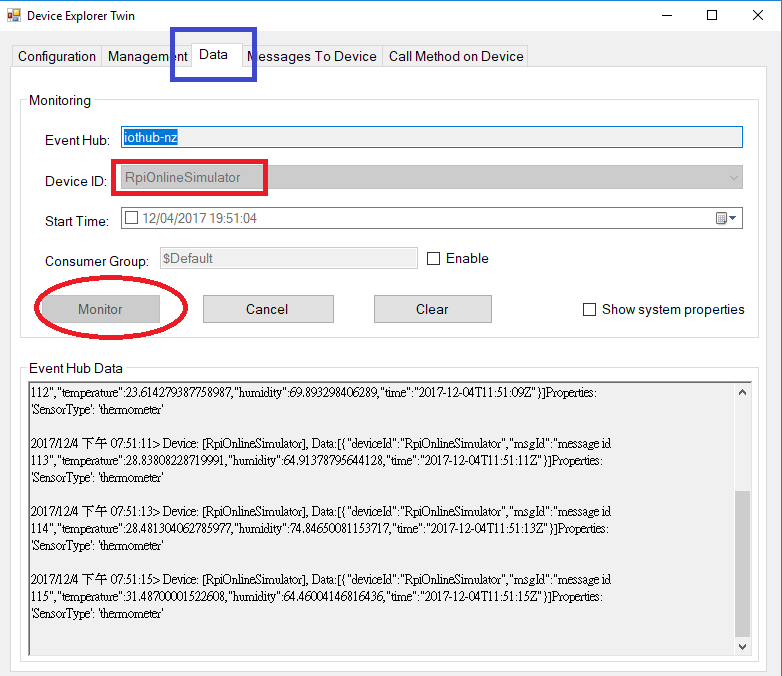
* + Done. Let’s start to run it. (You can set interval to 5000 ms if your IoT Hub is free tier)



* + Sending the D2C message every 5 seconds.



* Monitor the D2C messages in Device Explorer Tool.



* You can use the same C2D commands as above to test the Device-to-Cloud (C2D) messages.

(see the **c2d-command.txt** under **Scripts** folder)

{"command": "TURN\_ONOFF", "value":"1", "time":"2017-11-29T10:10:10.123Z"}

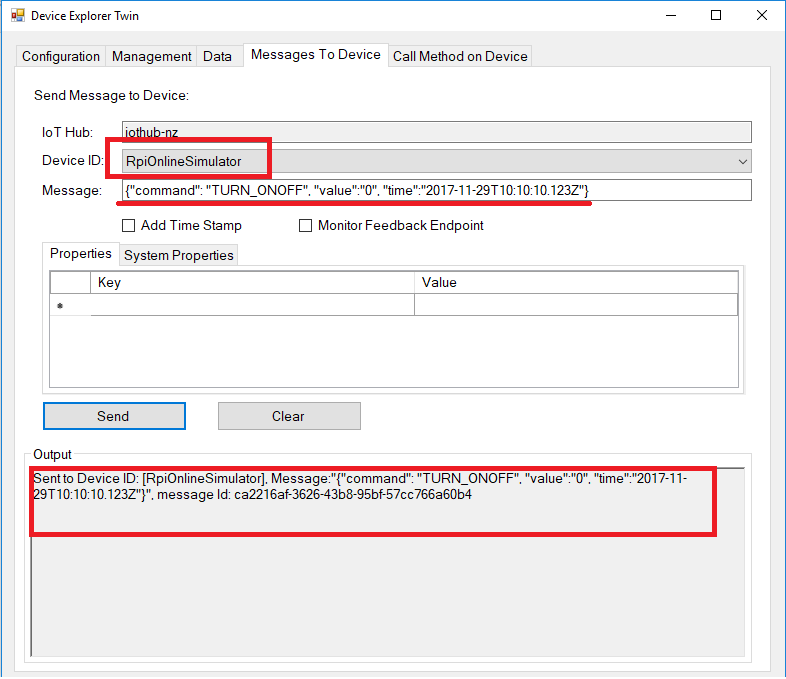
or

{"command": "TURN\_ONOFF", "value":"0", "time":"2017-11-29T10:10:10.123Z"}

or

{"command": "TEMPERATURE\_ALERT", "value":"50", "time":"2017-11-29T10:10:10.123Z"}

* + Select the **RpiOnlineSimulator** device.





* *The HOL 1 has been completed. Now you have one C# and one Node.js simulated devices which can be connected to Azure IoT Hub. They can send the telemetry data to cloud and receive the messages from cloud. In the next hands-on lab, we will learn how to process the IoT data as historic data by Azure Stream Analytics.*