## 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

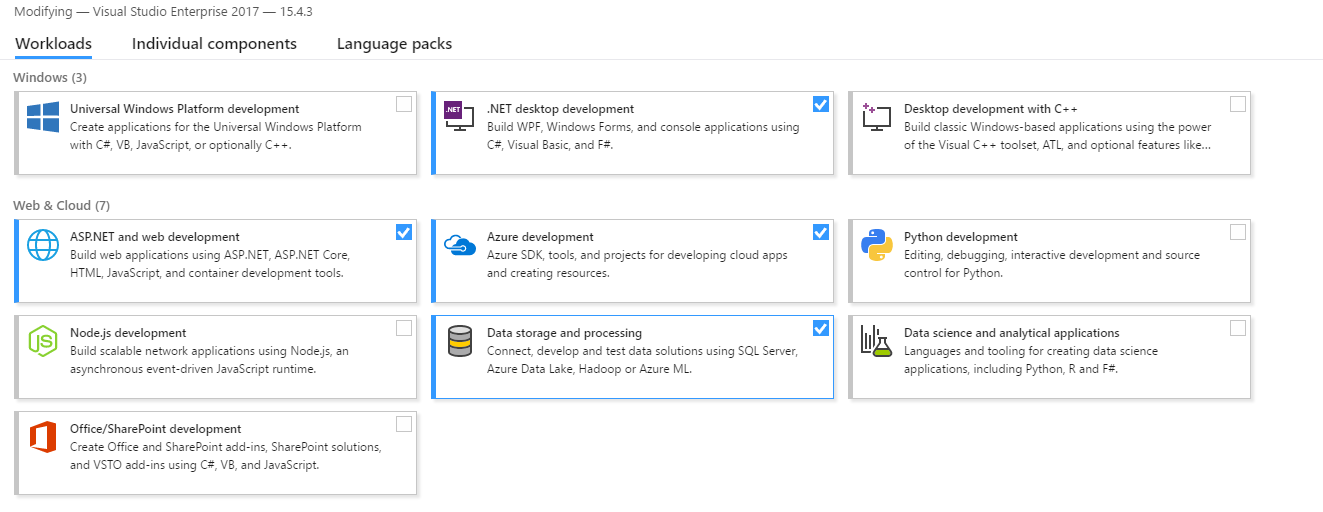
<https://github.com/Azure/azure-iot-sdks>

* 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 Visual Studio Code, and the extensions below:
  + Azure IoT Hub Extension
  + Azure Account (if not found)
* Microsoft Visual Studio 2017 with workloads
  + .NET desktop development
  + ASP.NET and web development
  + Azure Development
  + Data storage and processing

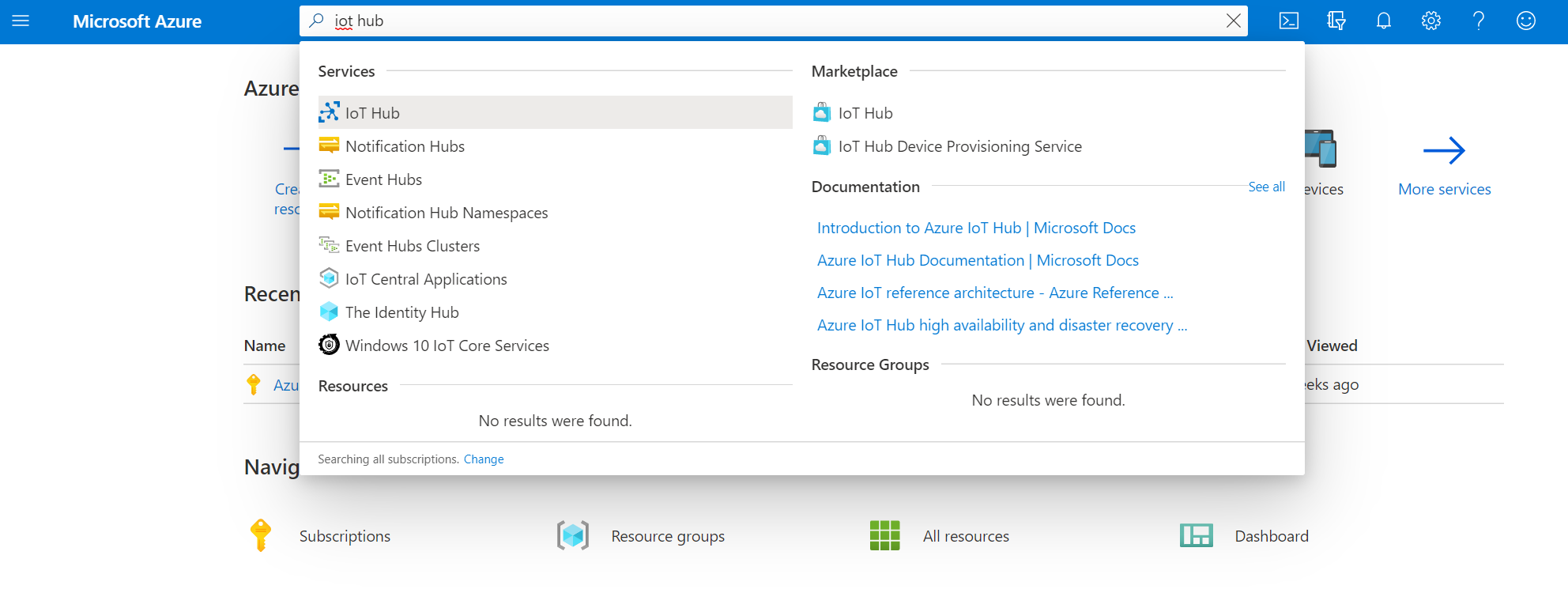


## Goals

* 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

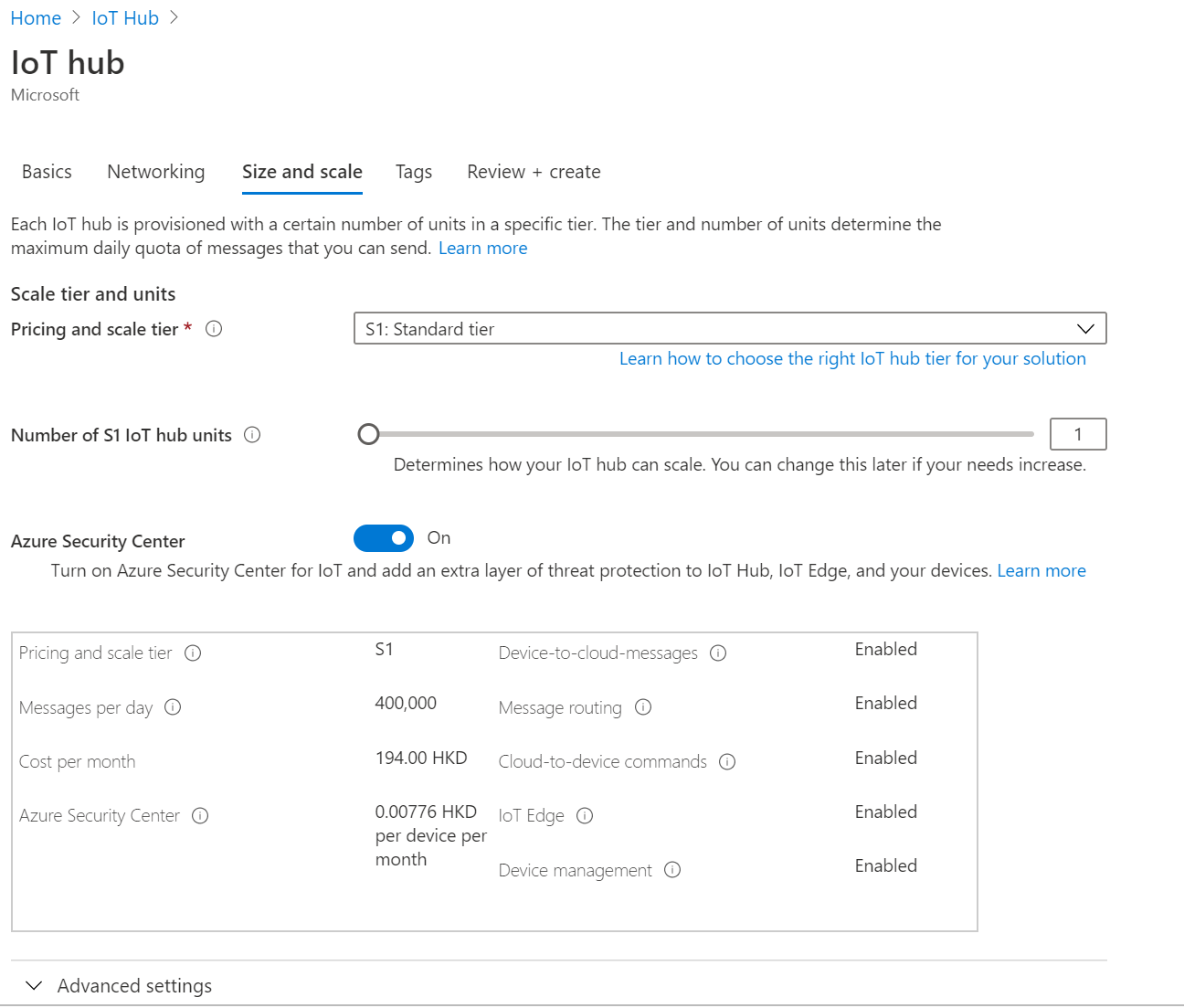
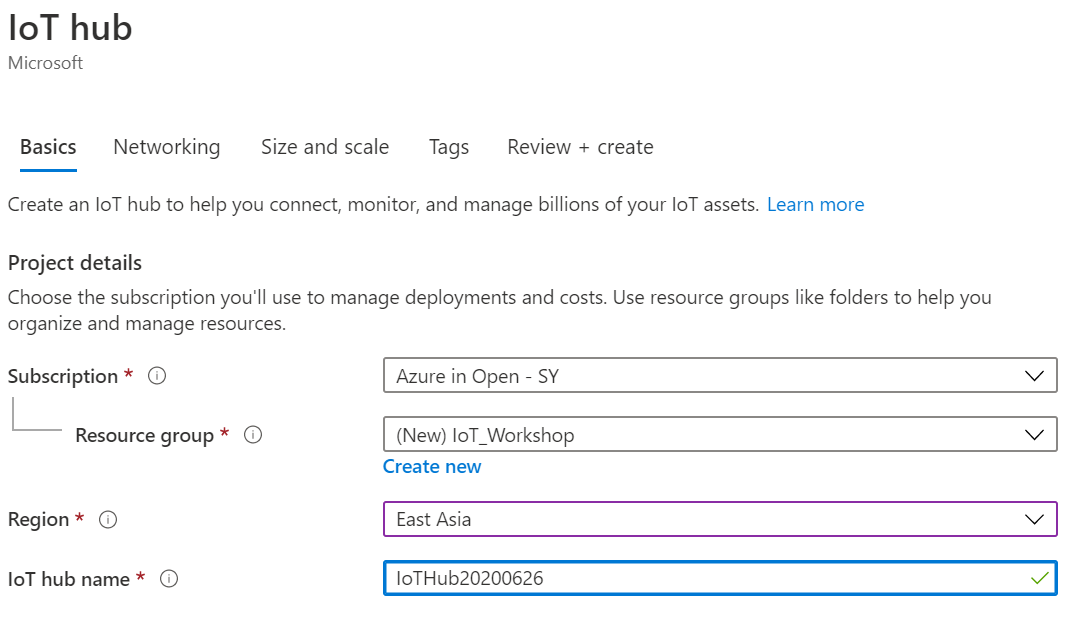
## Step 1: Provision an Azure IoT Hub

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



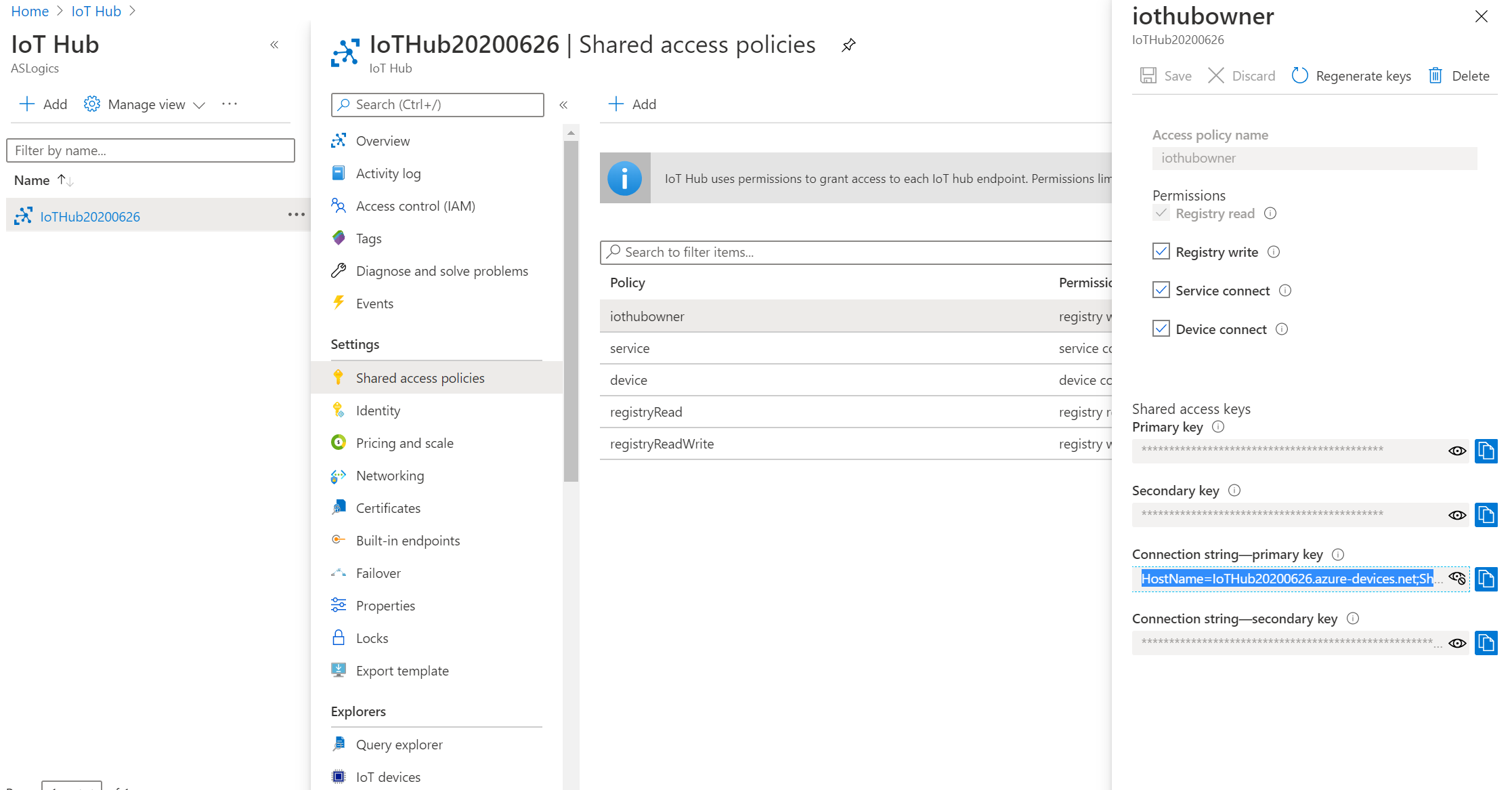
Here we select **S1 Standard** for this workshop

* + **Create** a new resource group to manage your solution.
  + Select a location (service region), for example, **East Asia**.
  + Ensure under Size and scale > Scale tier and units > Pricing and scaling tier, S1: Standard tier is selected
  + Leave other settings as it is

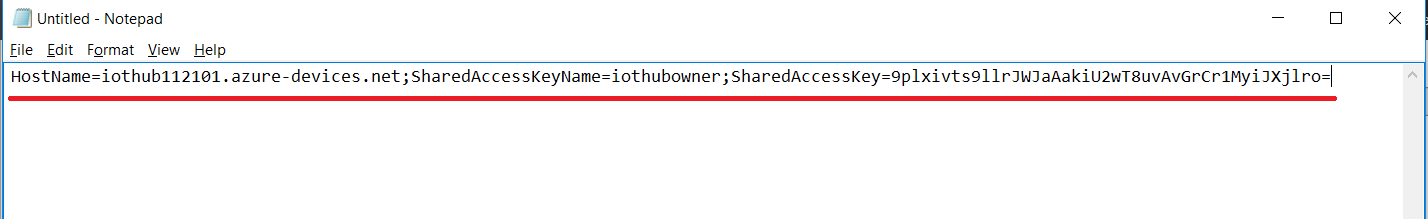


(It may take a few minutes to deploy IoT Hub)

* 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**

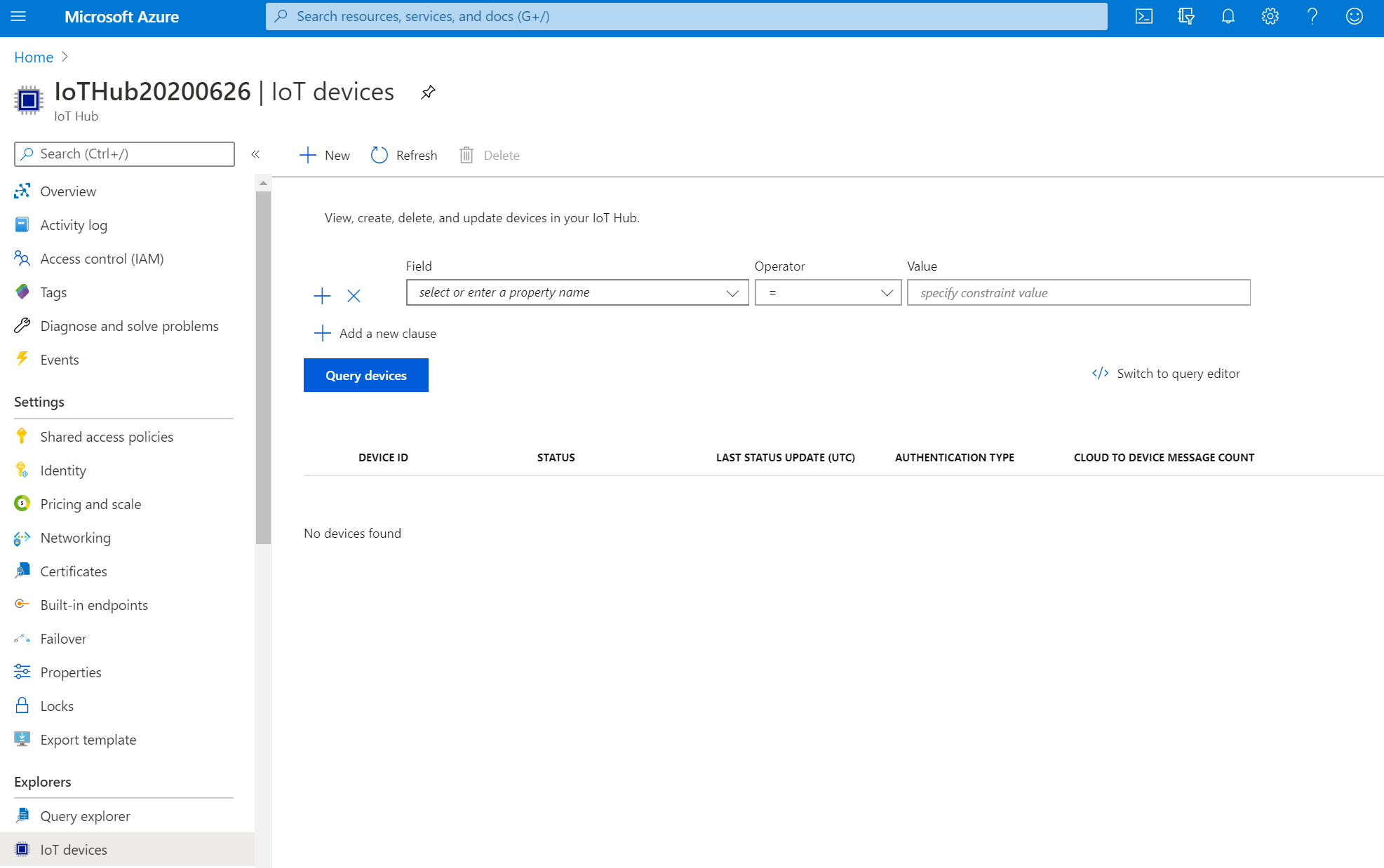


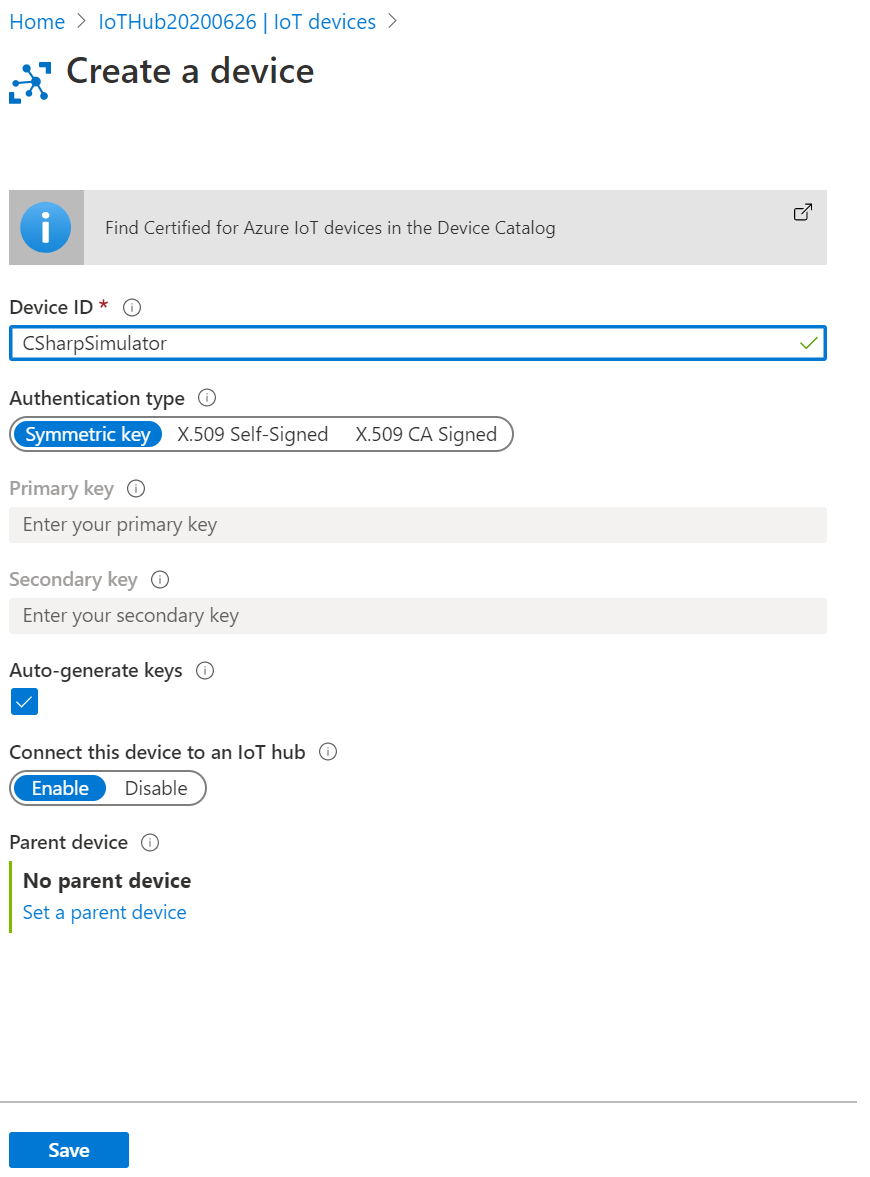
* + Save the **Connection String - primary key** for the later used.

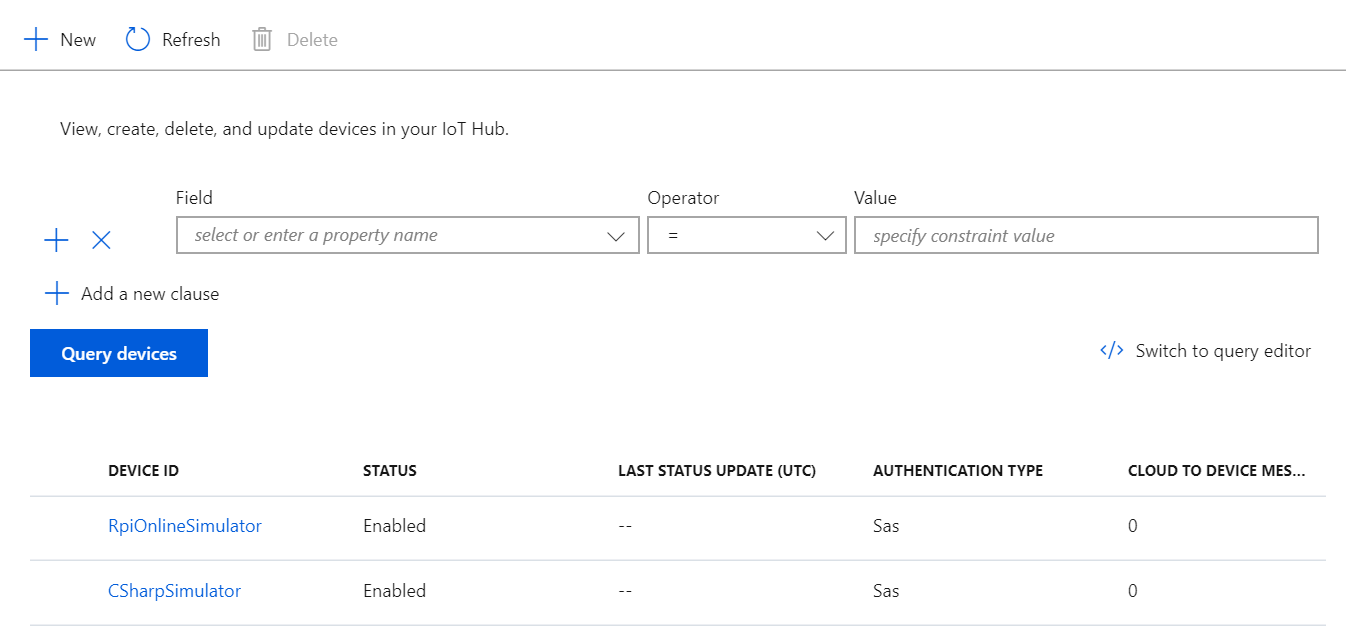


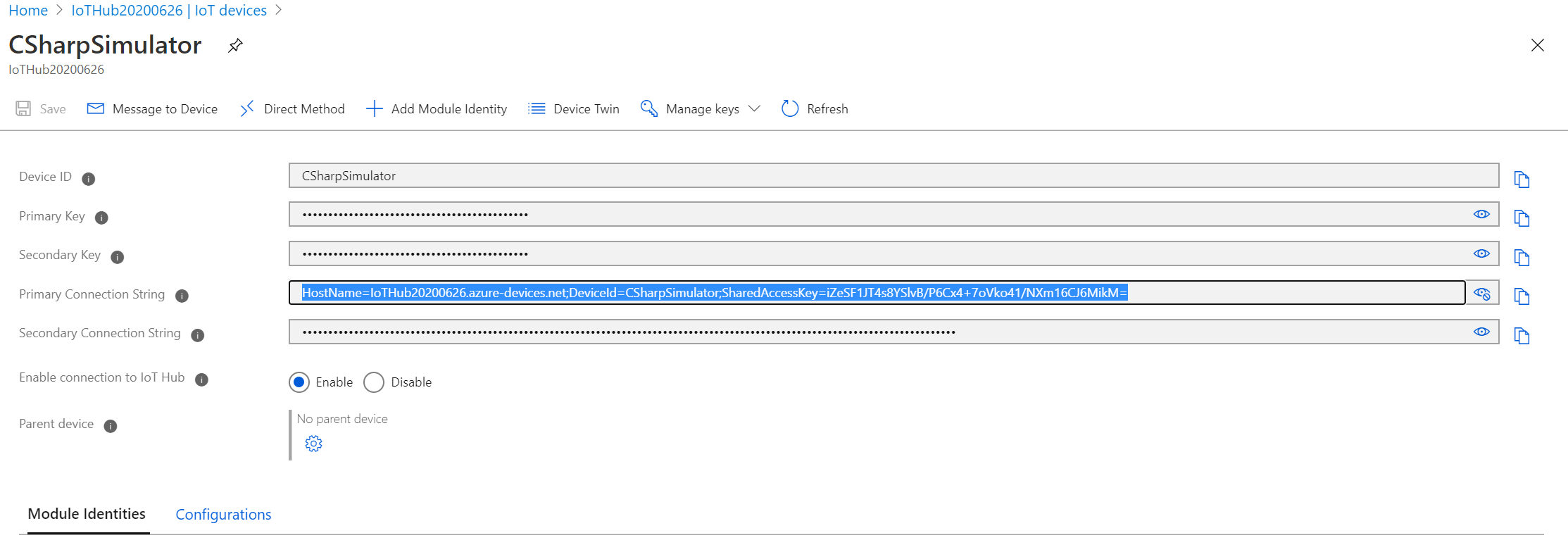
## Step 2: Provision 2 devices on IoT Hub

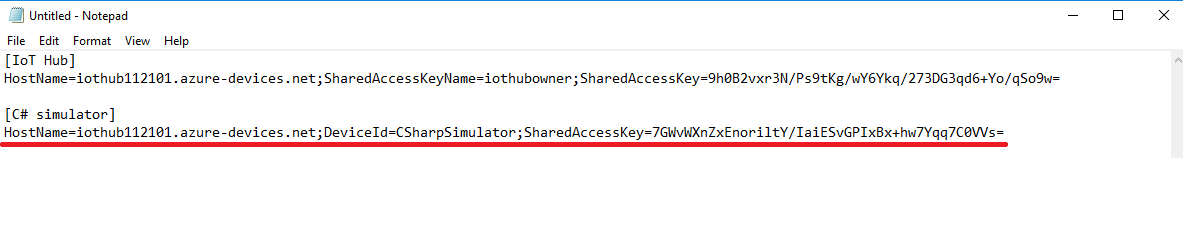
* In this IoT hub, and get the connection string of device.

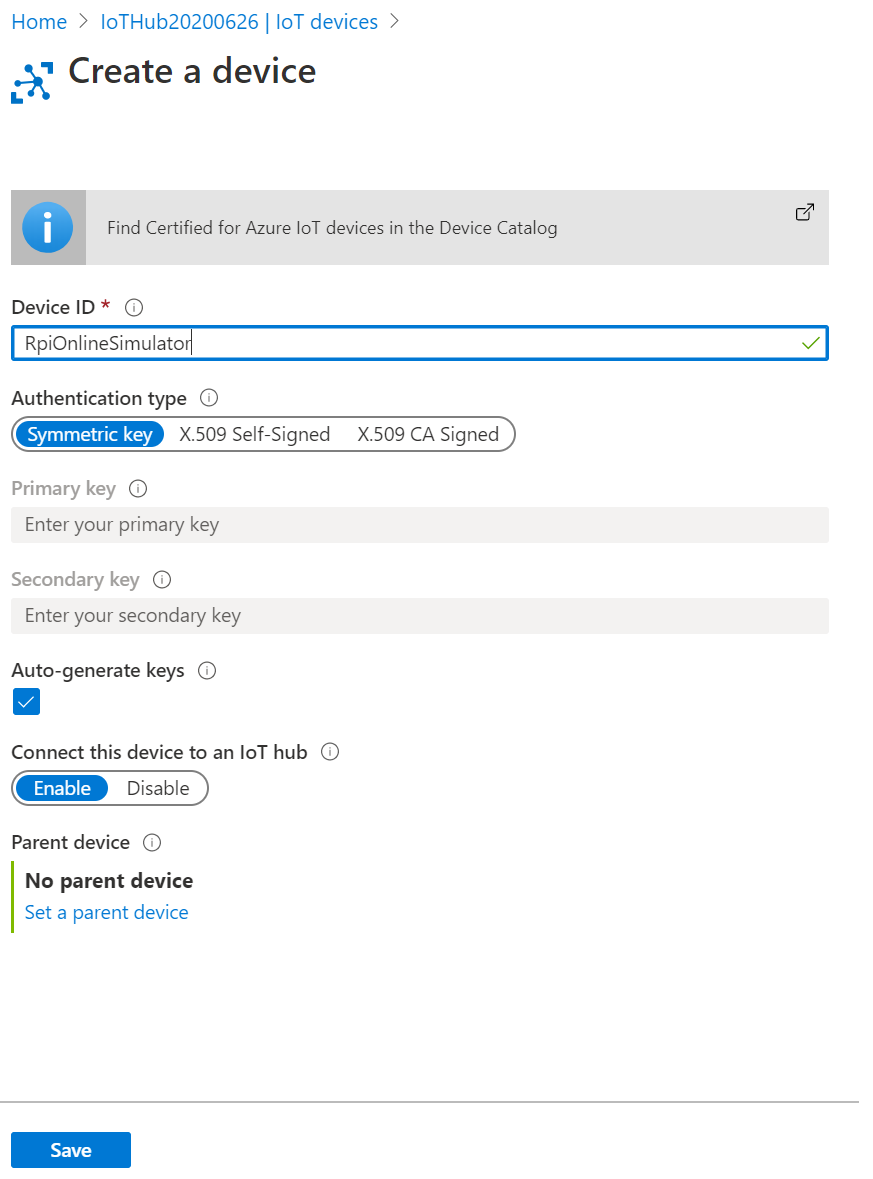


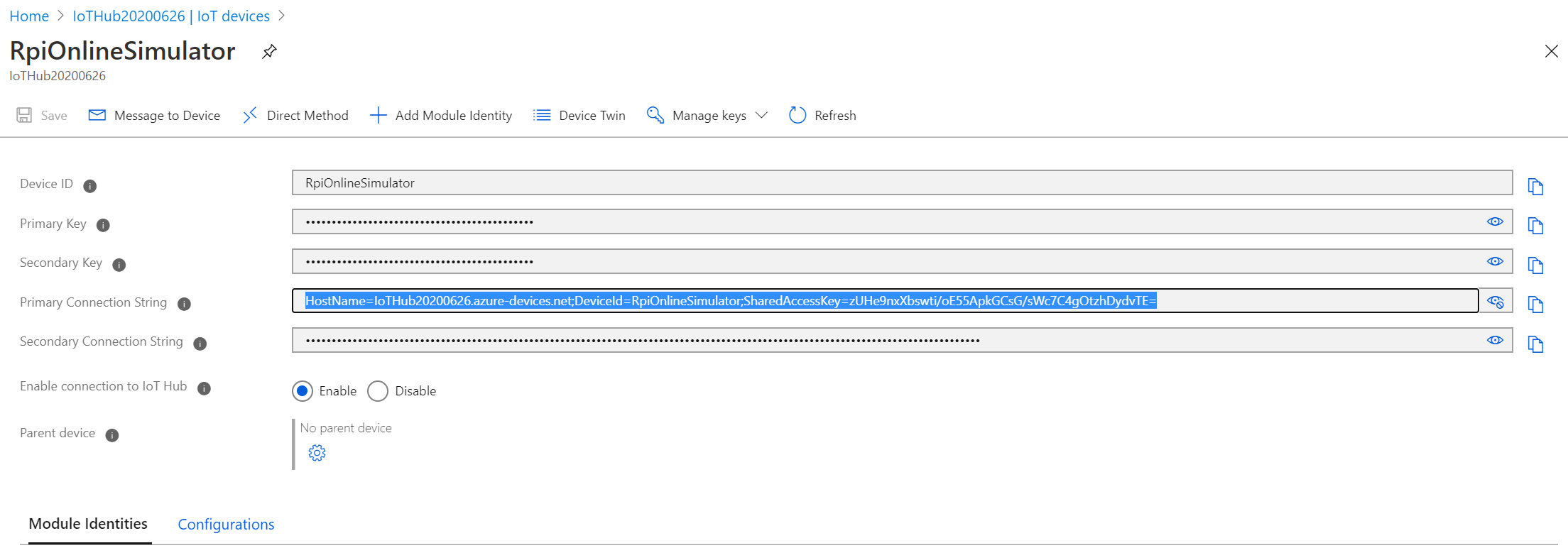
* **Create** the first simulated device, and named it to **CSharpSimulator**. (depends on your favorite name  
  
  + Created successfully

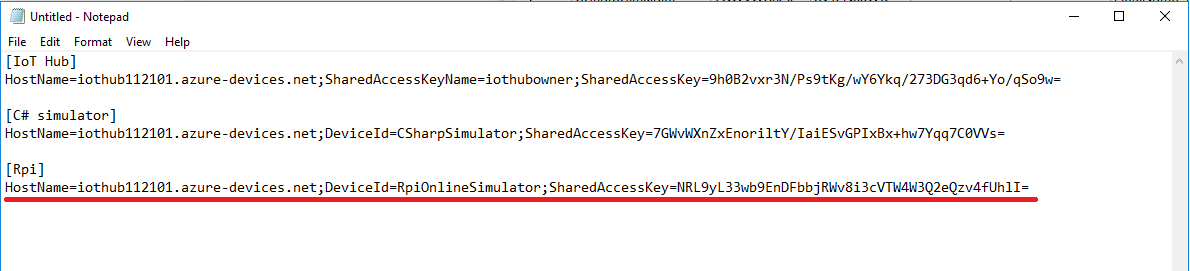


* When the list was updated, we **select** the **CSharpSimulator** and right-click to copy the connection string of device
  + Take a note with this connection string of device, and it will be used later.



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

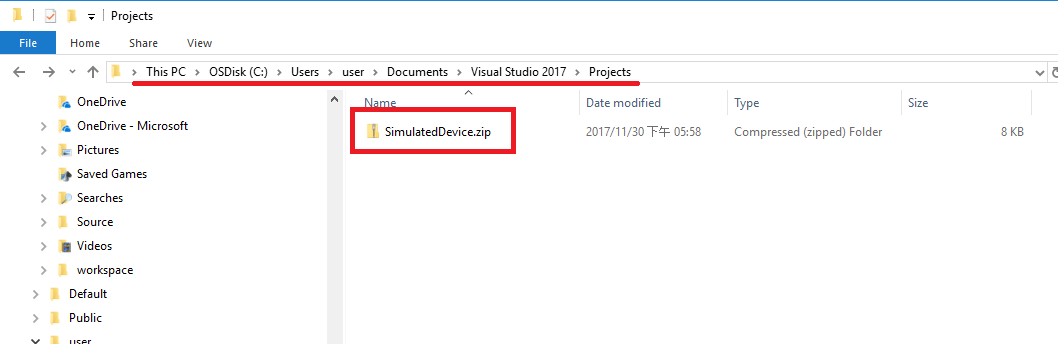




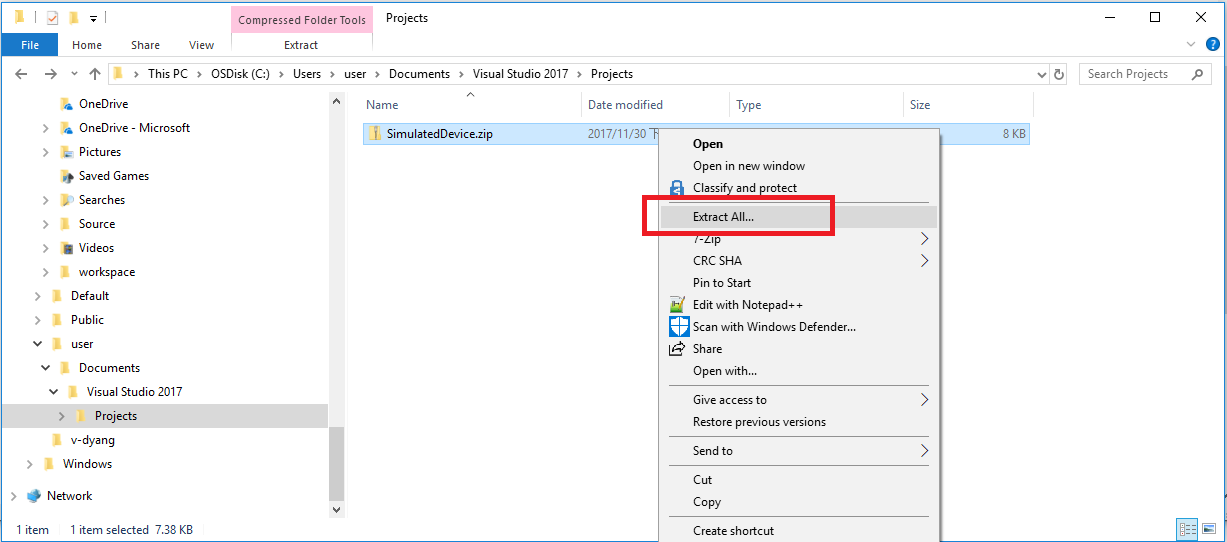
## Step 3: Build the C# simulated device

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

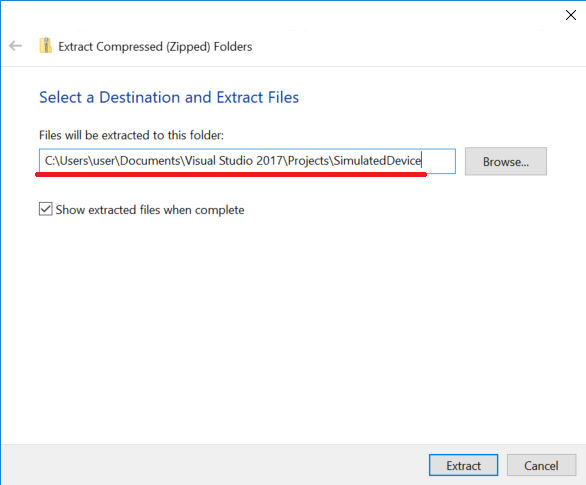
Path: **C:\Users\<username>\Documents\Visual Studio 2017\Projects\**



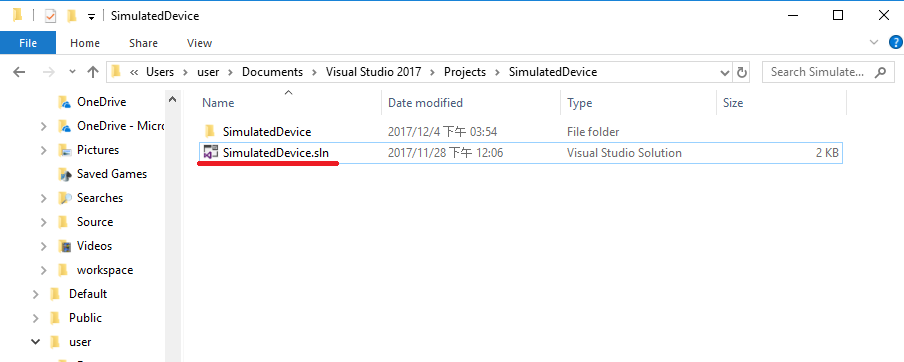
* Unzip



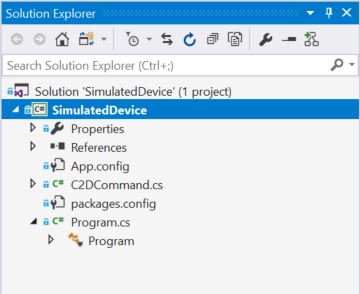
* The files should be extracted to **C:\Users\<username>\Documents\Visual Studio 2017\Projects\**

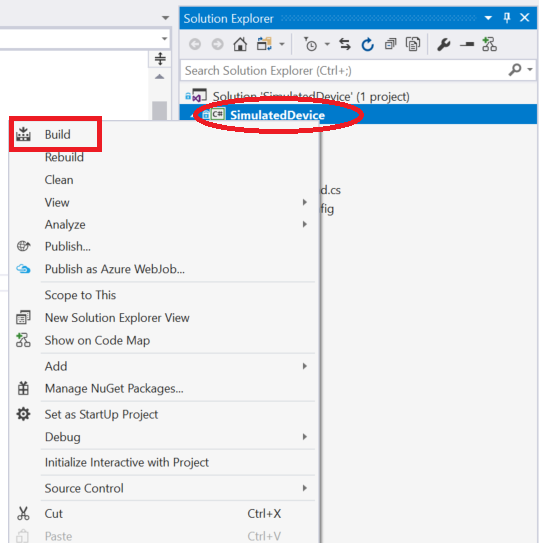


* Open the **SimulatedDevice** solution (.sln file) of VS

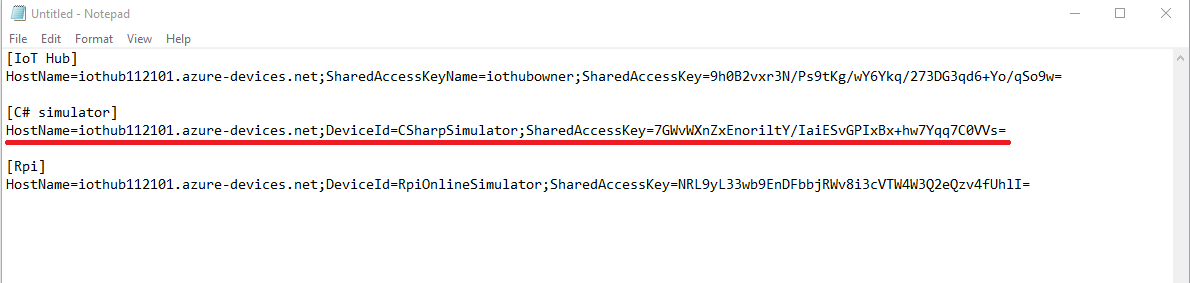


* Build the C# simulator
  + Right-click to build the project to restore all NuGet packages.

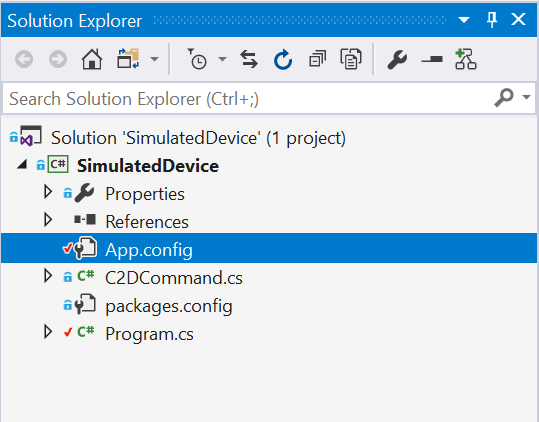


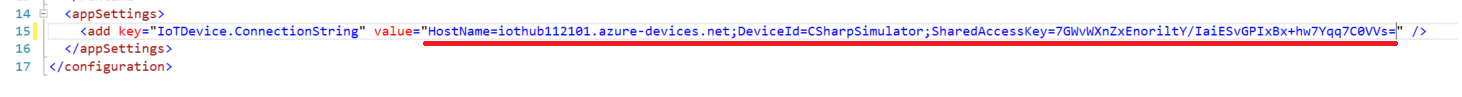


* + Get the device connection string



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



****

* + Open the **Program.cs**, and add the following code in line 17 to load the connection string from AppSettings.

string deviceConnectionString = ConfigurationManager.AppSettings["IoTDevice.ConnectionString"];

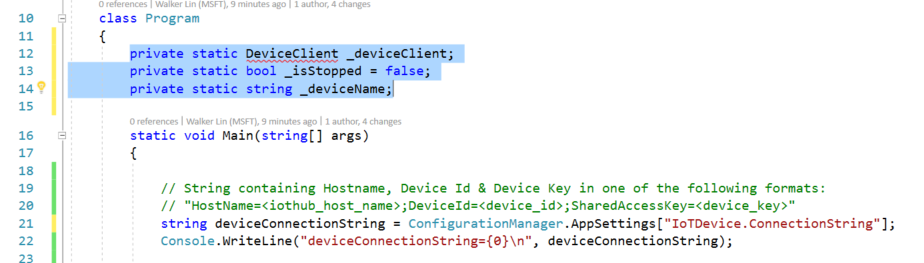


* + Add the global variables in Program class.

private static DeviceClient \_deviceClient;

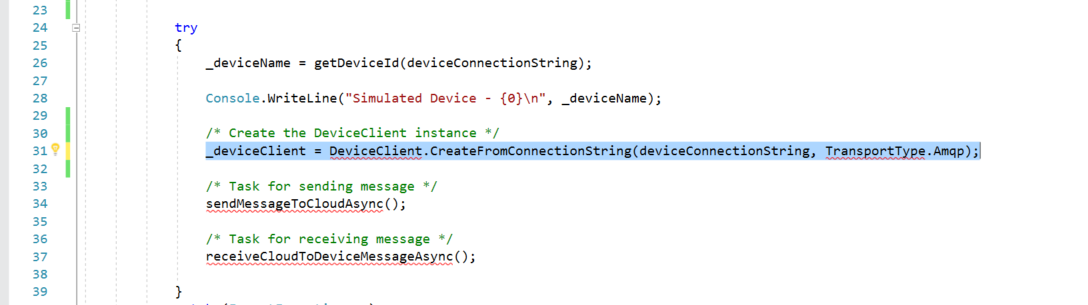
private static bool \_isStopped = false;

private static string \_deviceName;



* + Create a DeviceClient instance from Azure IoT Hub C# SDK.

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



* + Implement the **sendMessageToCloudAsync** method. First, please fill the telemetry data format in **telemetryDataPoint**.

var telemetryDataPoint = new

{

deviceId = \_deviceName,

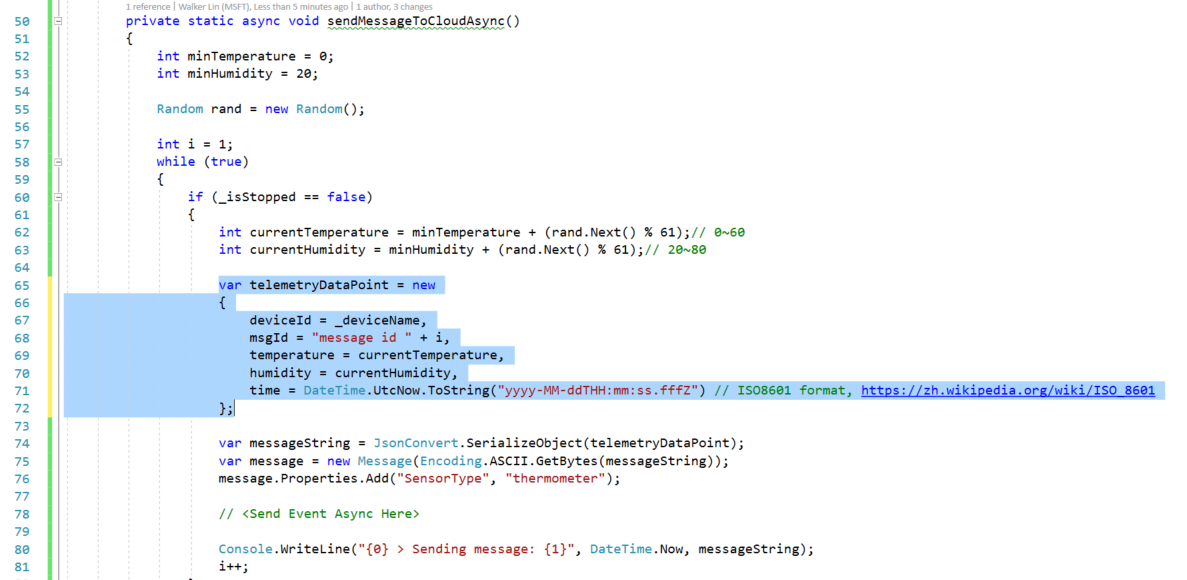
msgId = "message id " + i,

temperature = currentTemperature,

humidity = currentHumidity,

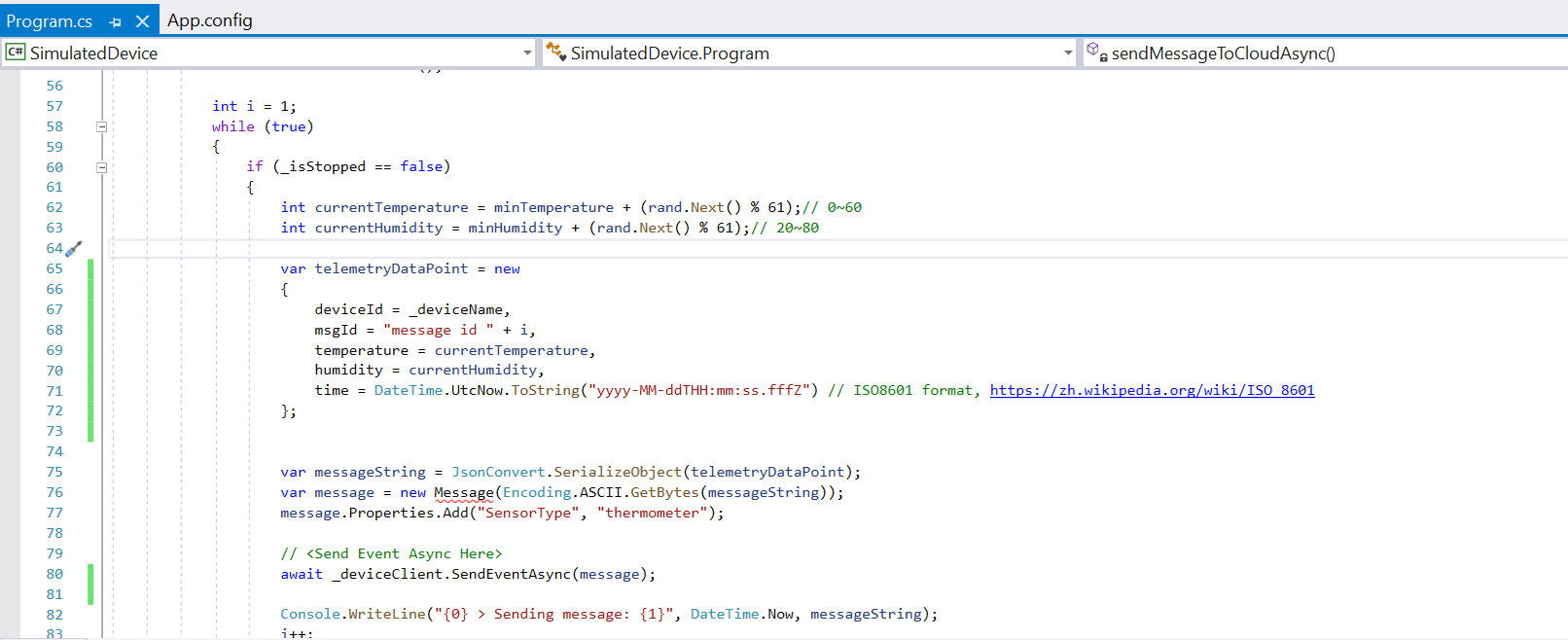
time = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ss.fffZ") // ISO8601 format, https://zh.wikipedia.org/wiki/ISO\_8601

};



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

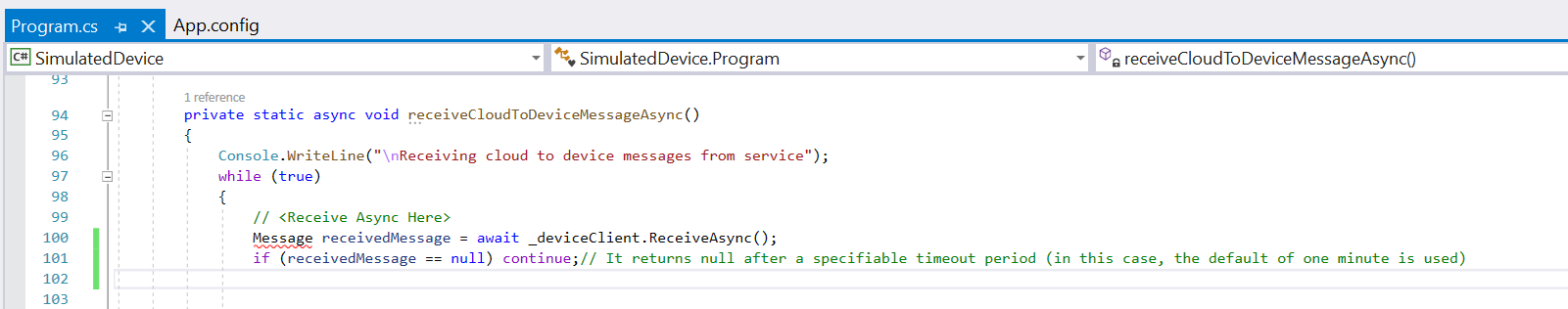
await \_deviceClient.SendEventAsync(message);



* + 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)



* + Implement **processCommand** method to process C2D command.

private static void processCommand(C2DCommand c2dCommand)

{

switch (c2dCommand.command)

{

case C2DCommand.COMMAND\_TEMPERATURE\_ALERT:

displayReceivedCommand(c2dCommand, ConsoleColor.Yellow);

break;

case C2DCommand.COMMAND\_TURN\_ONOFF:

displayReceivedCommand(c2dCommand, ConsoleColor.Green);

\_isStopped = c2dCommand.value.Equals("0"); // 0 means turn the machine off, otherwise is turning on.

break;

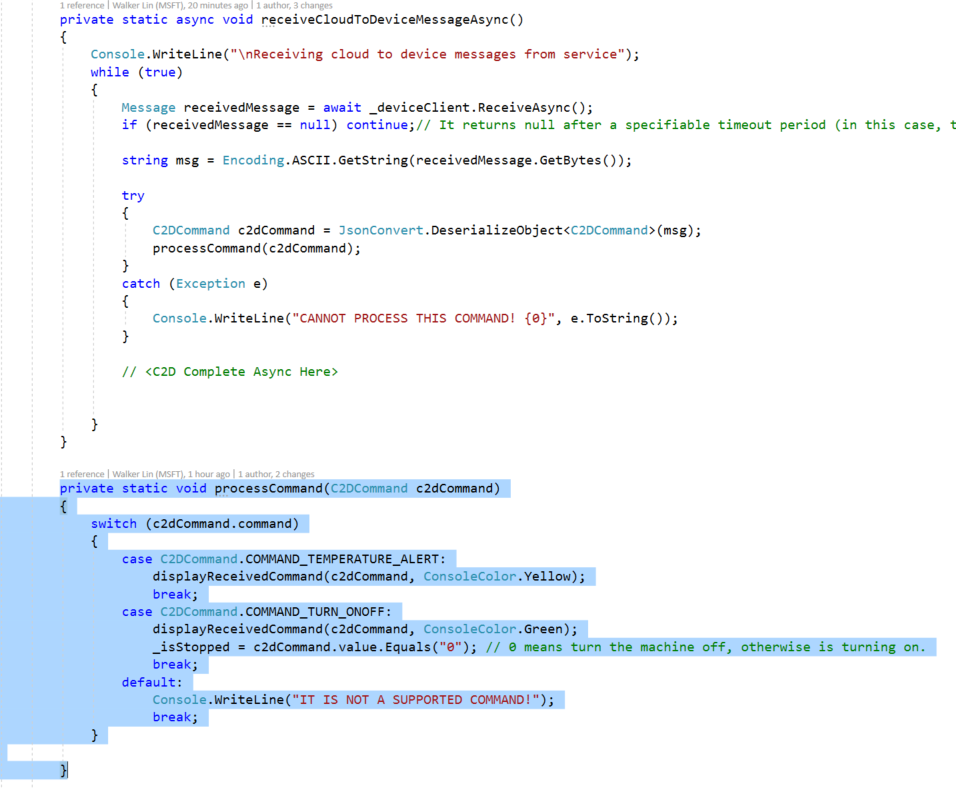
default:

Console.WriteLine("IT IS NOT A SUPPORTED COMMAND!");

break;

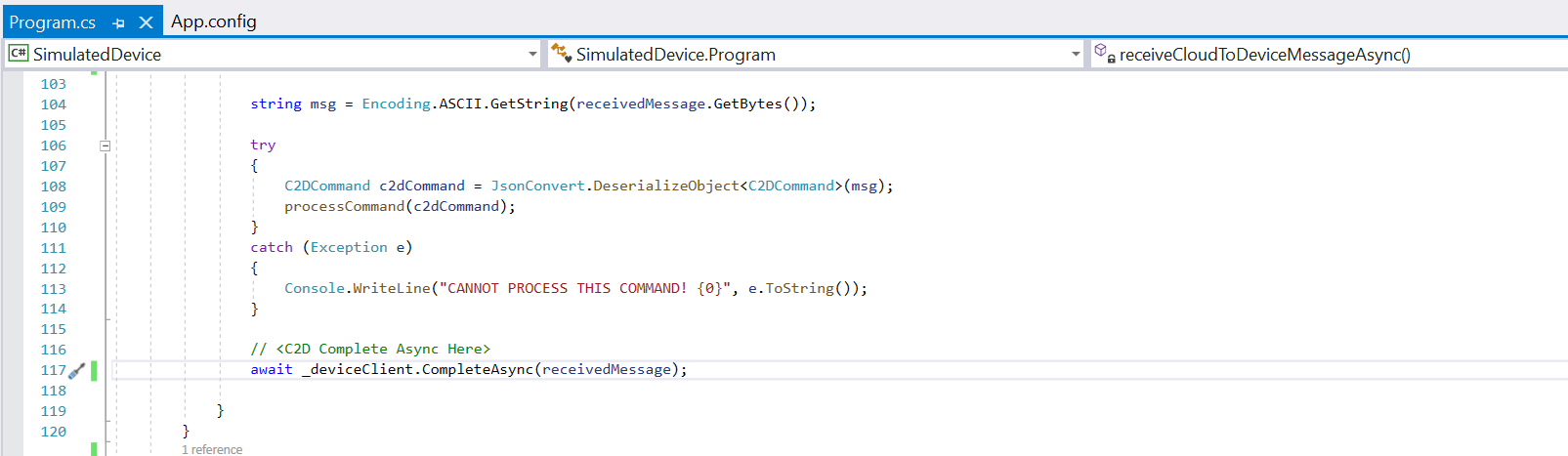
}

}

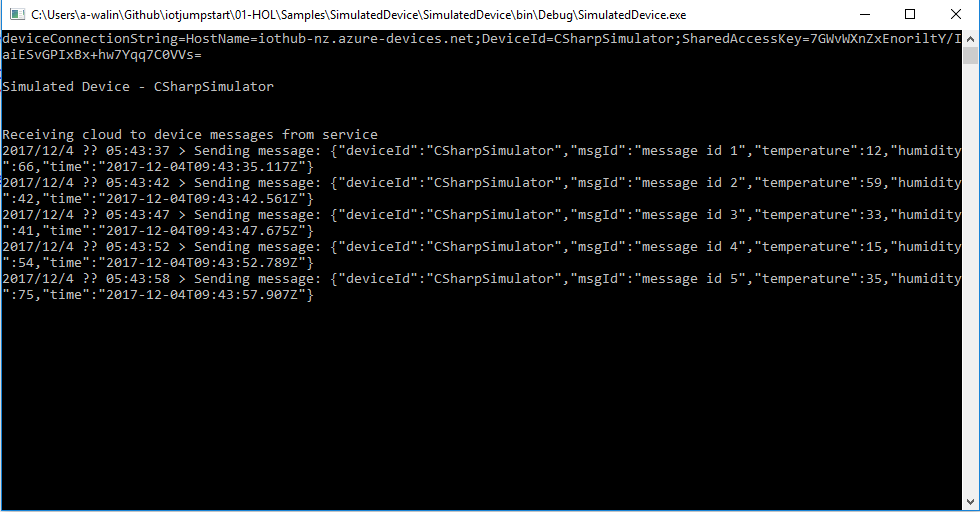


* + Complete the C2D message to IoT Hub.

await \_deviceClient.CompleteAsync(receivedMessage);



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

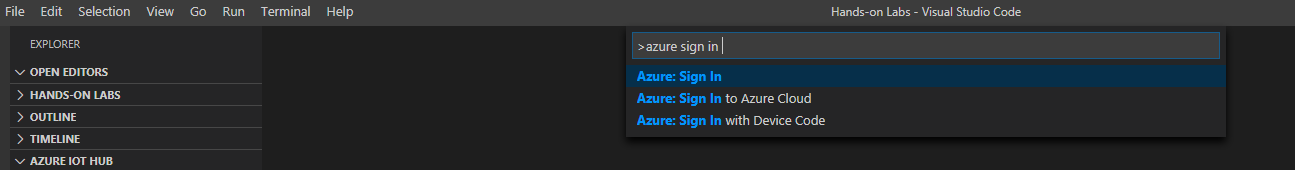


(Send the telemetry data every 5 seconds)

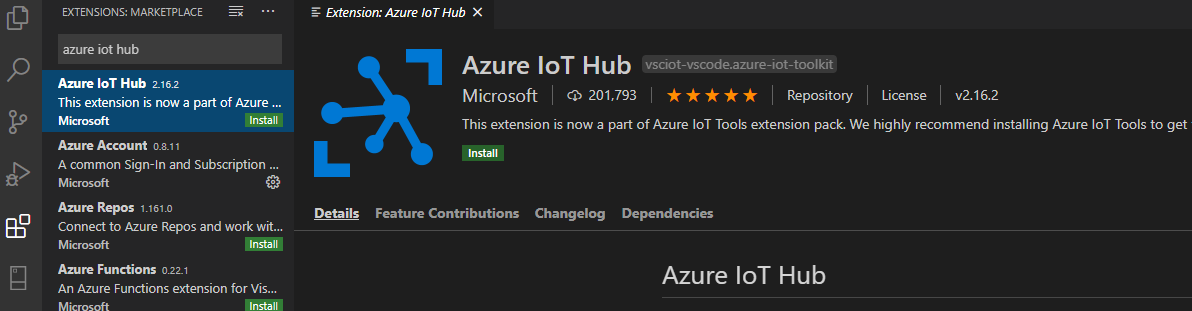
## Step 4: Observe the data communication between device and cloud

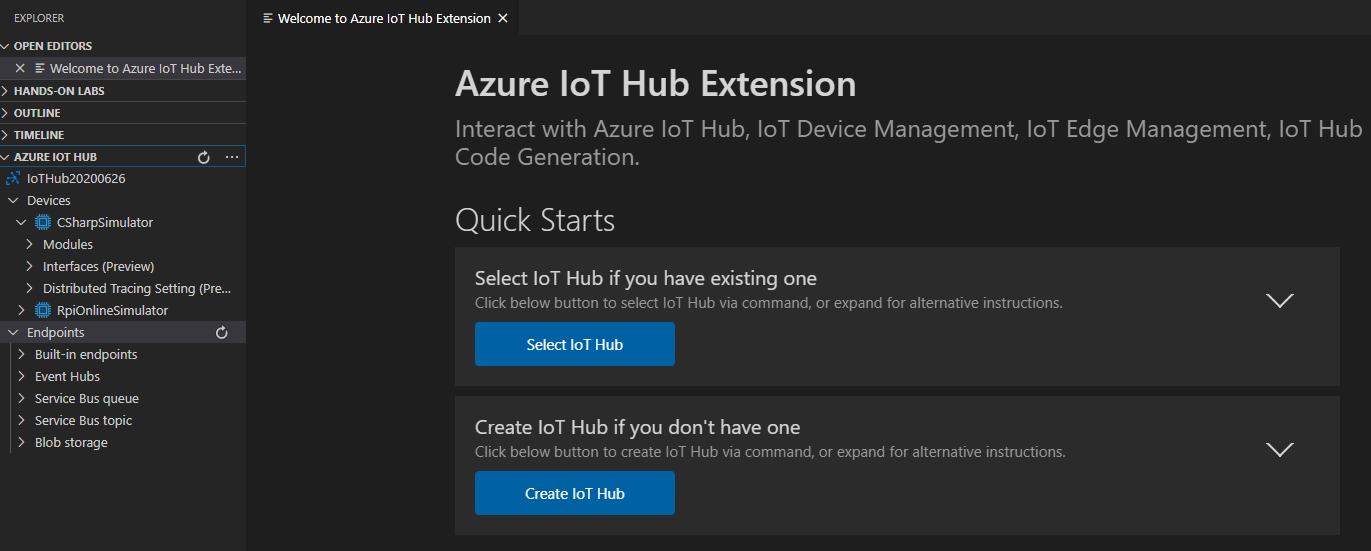
* In Visual Studio Code, login to your Azure account by activating



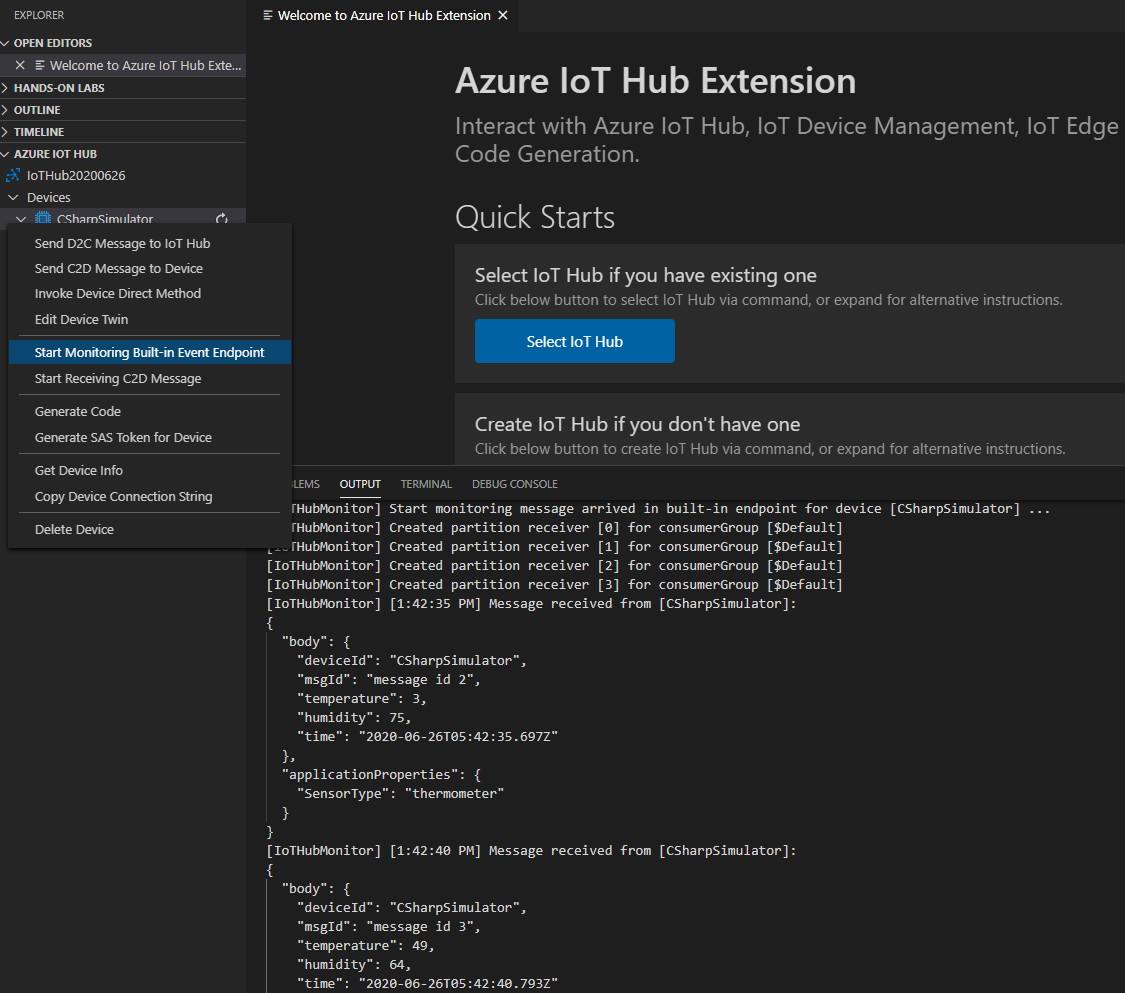


* Enable Azure IoT Hub Extension





* Right click CSharpSimulator and click ‘Start Monitoring Built-in Event Endpoint



**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.**

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

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

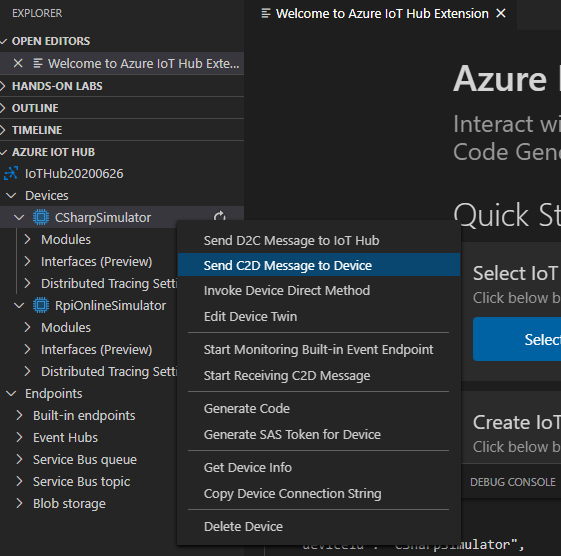
{"command": "TURN\_ONOFF", "value":"1", "time":"2017-11-29T10:10:10.123Z"}

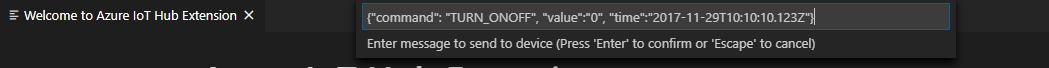
or

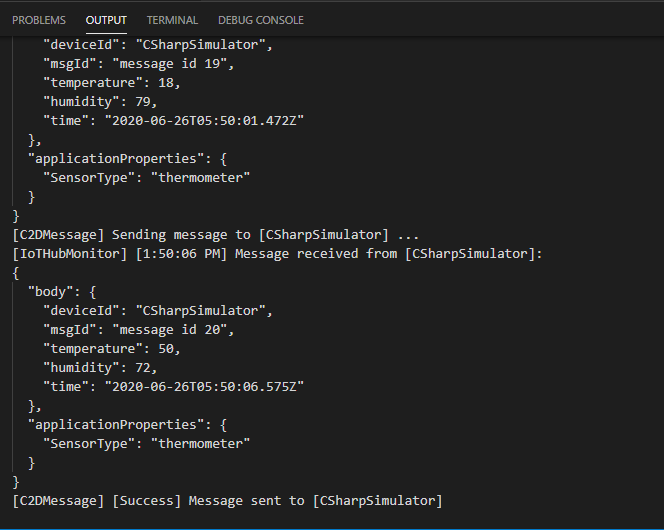
{"command": "TURN\_ONOFF", "value":"0", "time":"2017-11-29T10:10:10.123Z"}

or

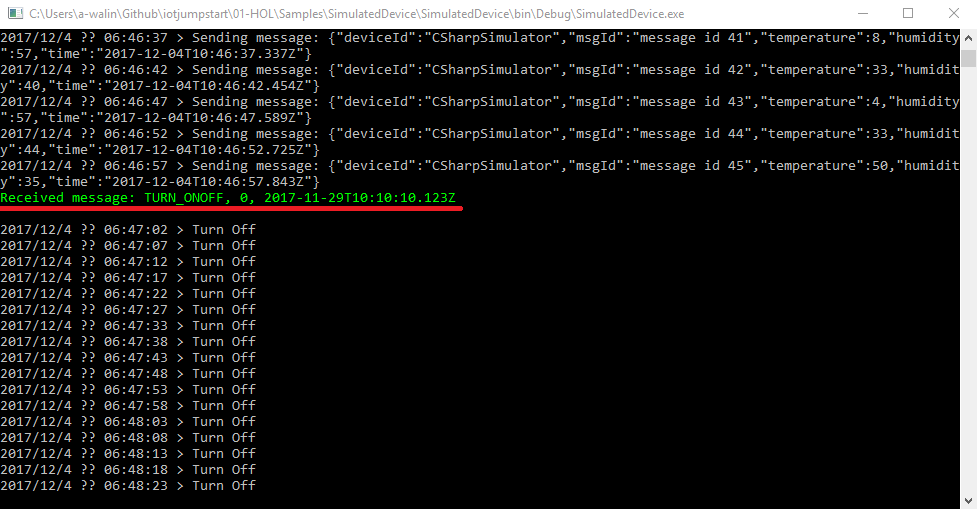
{"command": "TEMPERATURE\_ALERT", "value":"50", "time":"2017-11-29T10:10:10.123Z"}







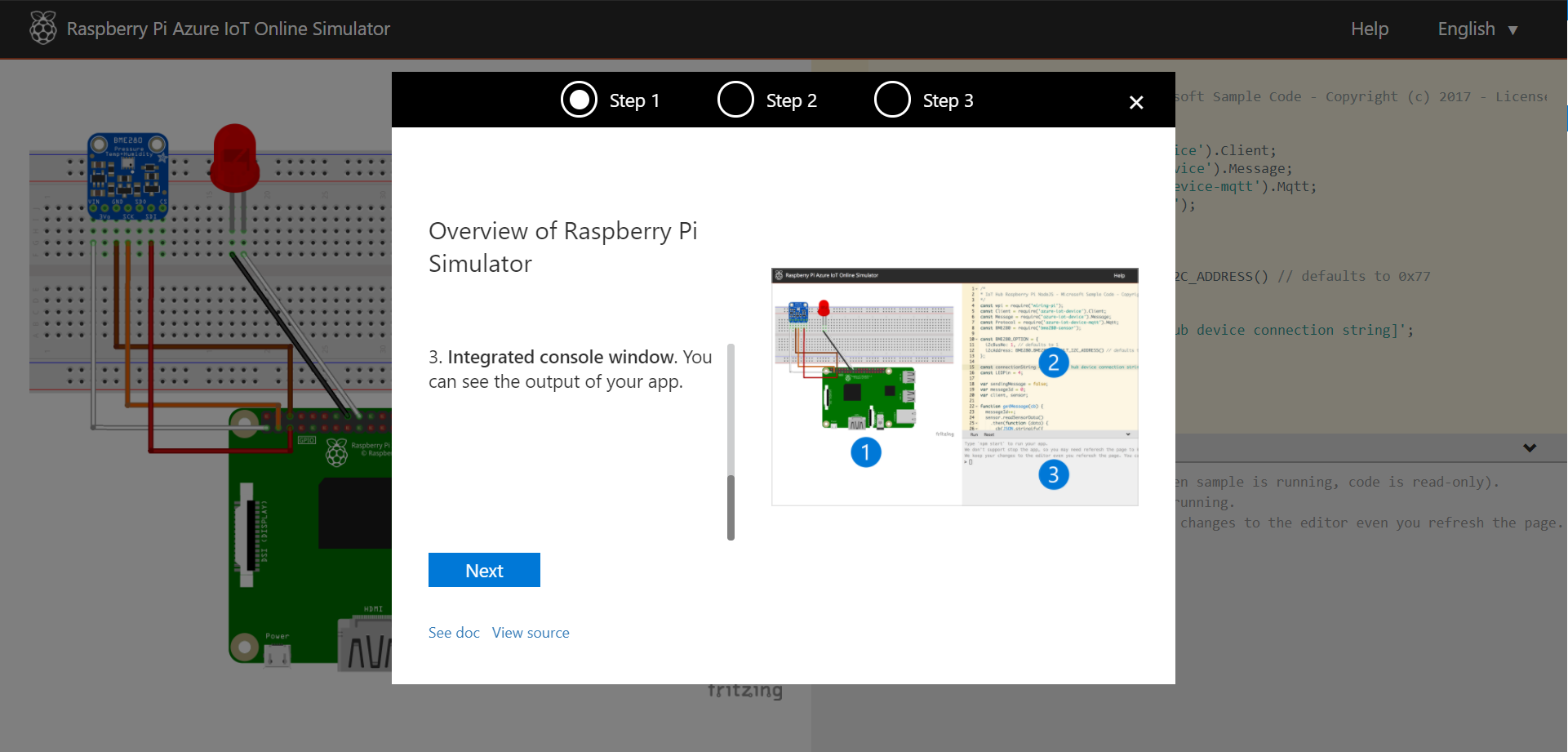
* The output of Windows Console App



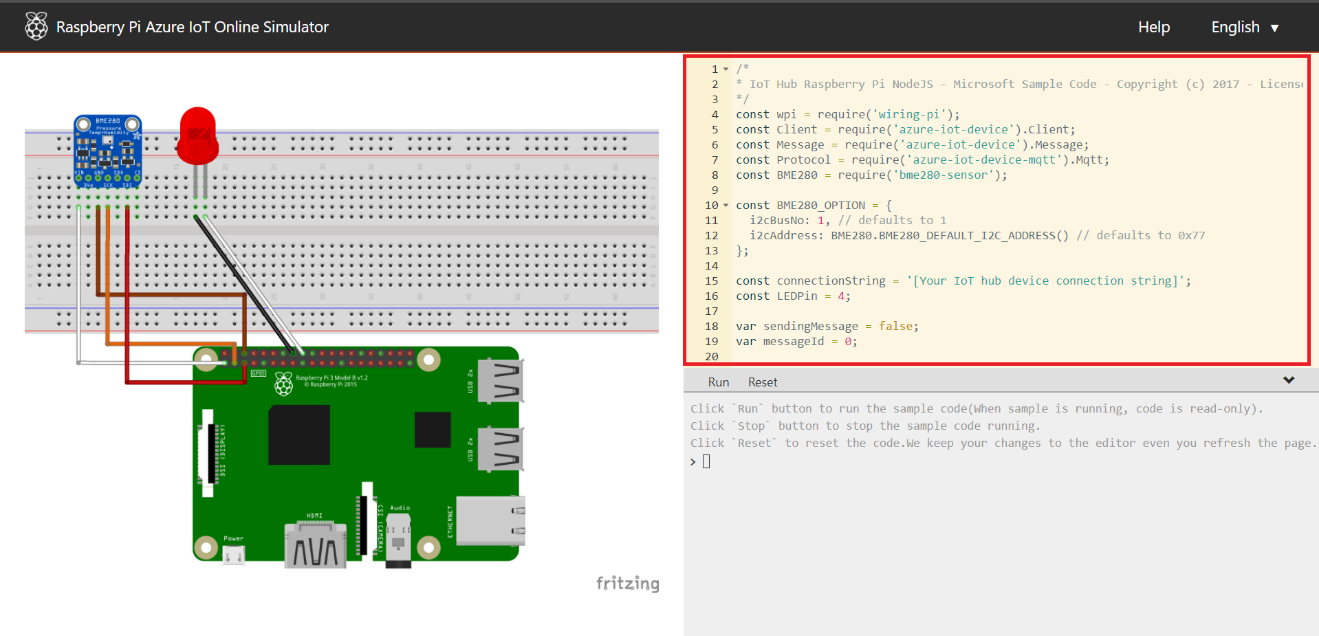
## Step 5: Run the second simulator on Raspberry Pi Azure Online Simulator

* Open this website on your browser

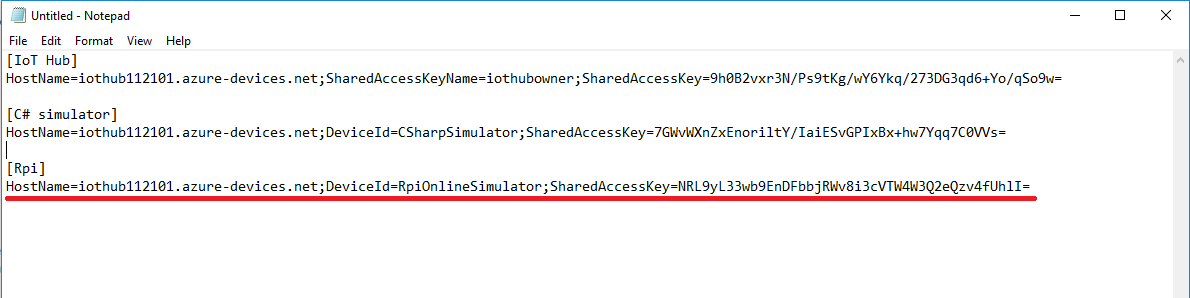
<https://azure-samples.github.io/raspberry-pi-web-simulator/>

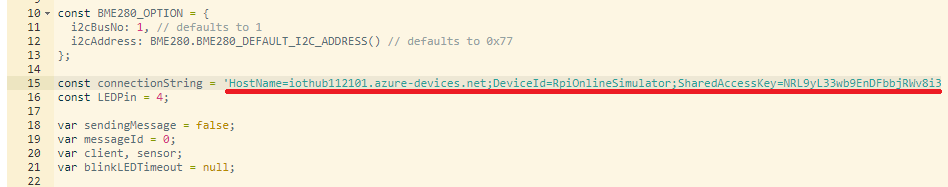


* Finish the tutorial, then you can start to write your Node.js code here.



* Coding now!
  + Put your device connection string in line 15.





* + Change the telemetry data format

function getMessage(cb) {

messageId++;

sensor.readSensorData()

.then(function (data) {

cb(JSON.stringify({

deviceId: getDeviceId(connectionString),

msgId: 'message id ' + messageId,

temperature: data.temperature\_C,

humidity: data.humidity,

time: getUTCTime()

}));

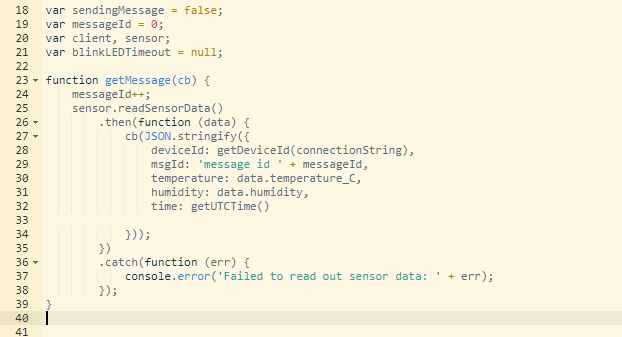
})

.catch(function (err) {

console.error('Failed to read out sensor data: ' + err);

});

}



* + Add the **getDeviceId** and **getUTCTime** functions

function getDeviceId(cs) {

var fields = cs.split(';');

return fields[1].substring(fields[1].indexOf('=') + 1);

}

function getUTCTime() {

return new Date().toISOString().

replace(/\..+/, '') + "Z"; // delete the dot and everything after

}



* + Replace the **getMessage** function for **SensorType** property.

getMessage(function (content) {

var message = new Message(content);

message.properties.add('SensorType', 'thermometer');

console.log('Sending message: ' + content);

client.sendEvent(message, function (err) {

if (err) {

console.error('Failed to send message to Azure IoT Hub');

} else {

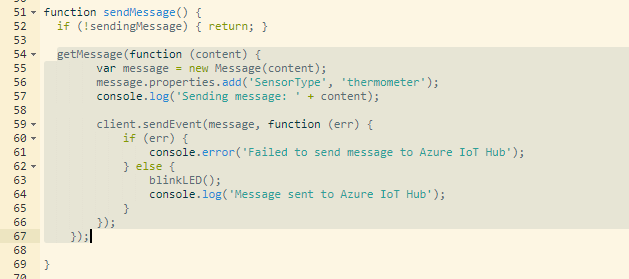
blinkLED();

console.log('Message sent to Azure IoT Hub');

}

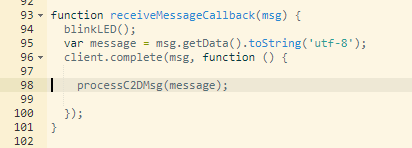
});

});



* + Add a function call for processing the C2D message.

processC2DMsg(message);



* + Implement the processC2DMsg function.

function processC2DMsg(message) {

try {

var c2dMsg = JSON.parse(message);

if (c2dMsg.command !== null) {

switch (c2dMsg.command) {

case 'TEMPERATURE\_ALERT':

console.log(c2dMsg.time + '>>>>> TEMPERATURE\_ALERT: ' + c2dMsg.value);

break;

case 'TURN\_ONOFF':

console.log(c2dMsg.time + '>>>>> TURN\_ONOFF: ' + c2dMsg.value);

if (c2dMsg.value === '0')

sendingMessage = false;// TURN OFF

else

sendingMessage = true;// TURN ON

break;

default:

printReceiveMessage(message);

break;

}

}

else

printReceiveMessage(message);

}

catch (e) {

printReceiveMessage(message);

}

}



* + The last one step. Add the **printReceiveMessage** function to show the C2D messages.

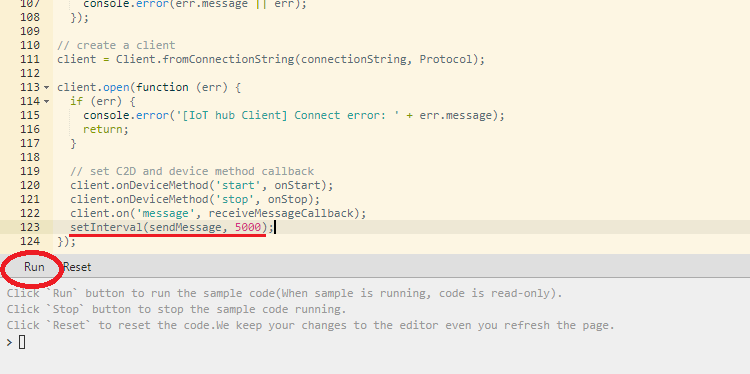
function printReceiveMessage(msg) {

console.log('>>>>> Receive message: ' + msg);

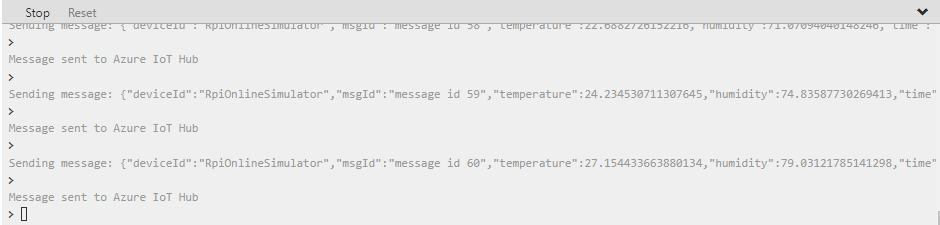
}



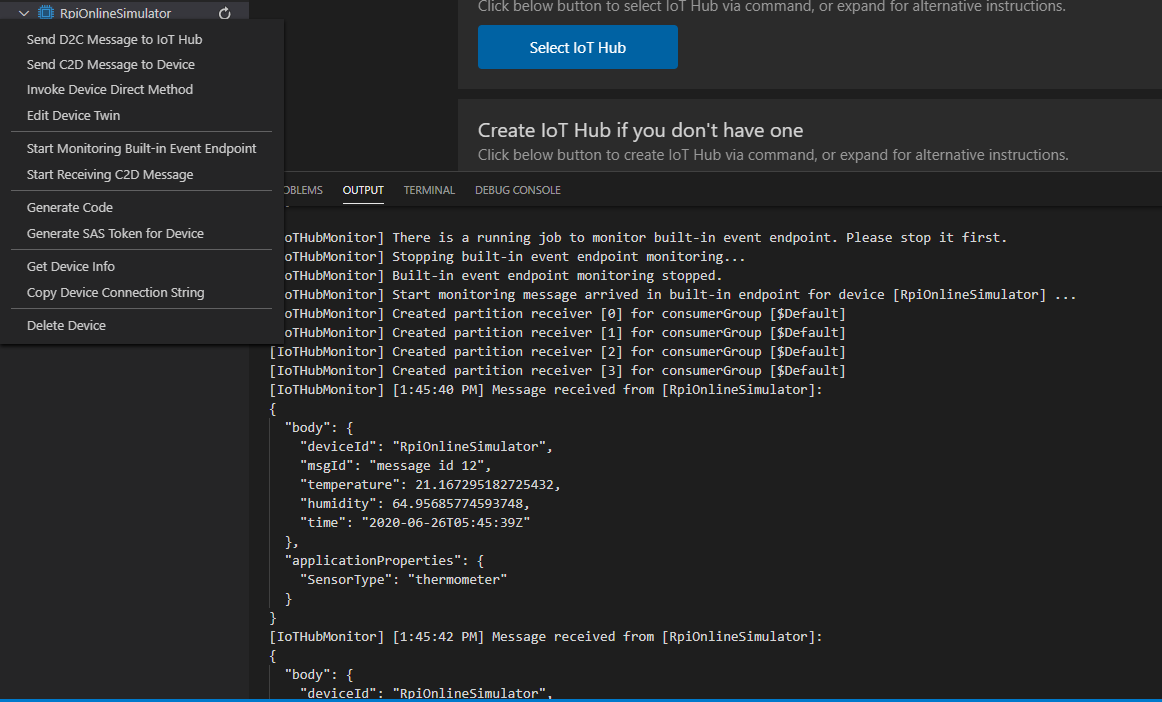
* + Done. Let’s start to run it. (You can set interval to 5000 ms if your IoT Hub is free tier)



* + Sending the D2C message every 2 seconds.



* Monitor the D2C messages in Device Explorer Tools.



* 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

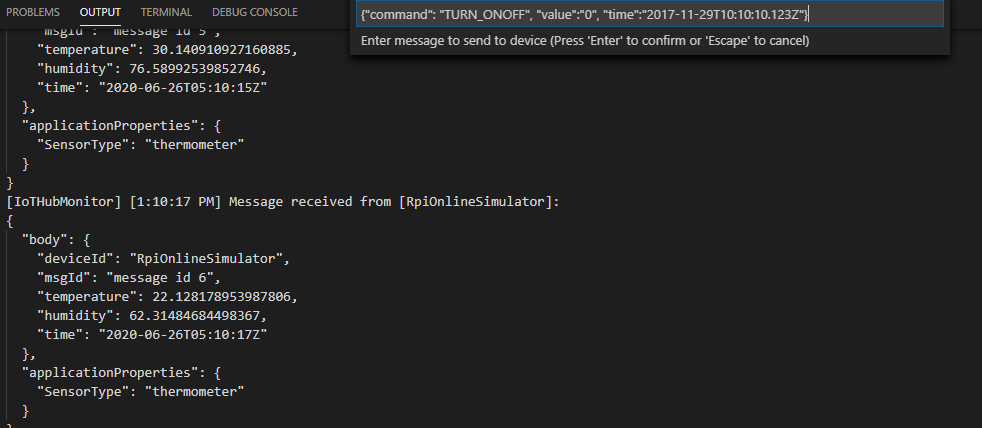
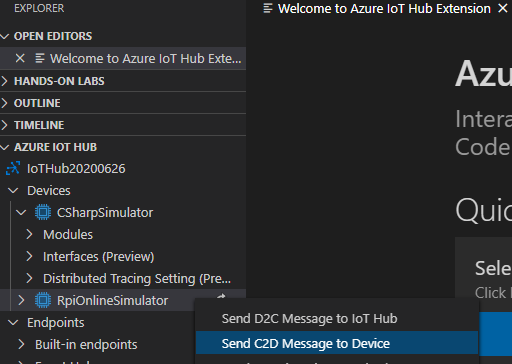
{"command": "TURN\_ONOFF", "value":"0", "time":"2017-11-29T10:10:10.123Z"}

or

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

* + Select the **RpiOnlineSimulator** device.

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





* *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.*