

Event Hub and IoT Hub

Lab 09

by

Andrea Hatch, Nishava Inc.

Deep Azure @McKesson

Overview

- I. Set-up all Prerequisites
- II. Event Hubs & demo
- III. IoT Hubs & demo

Objective of Demos

- The demos I will be showing will show how to work with messages
- With an Event Hub I will show to send a message using one console application
- With an IoT Hub I will show how to send messages and receive messages using multiple console applications

My Environment

- Windows 7
- Visual Studio 2017
- .Net Core
- Azure (free trial)

Visual Studio
Community 2017



I. Prerequisites

Prerequisites

- You will need to have the following before working with Event Hub messaging:
 - Visual Studio
 - .NET Core
 - Azure
 - Create an Event Hub Namespace in Azure
 - Create an Event Hub in Azure
- You will need to have the following before working with IoT Hub messaging:
 - Visual Studio
 - Azure
 - Create an IoT Hub in Azure

II. Event Hubs

Event Hubs

- Highly scalable data streaming platform
- Built to have high throughput while storing millions of events
- Able to receive and process millions of events every second
- They are built to be able to process both real time and batch process at the same time
 - This helps to limit the complexities of solutions making it an easy way to load data in Azure



Source: <https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-what-is-event-hubs>

Event Hubs SDKs

These are the only 2 programming options available today:

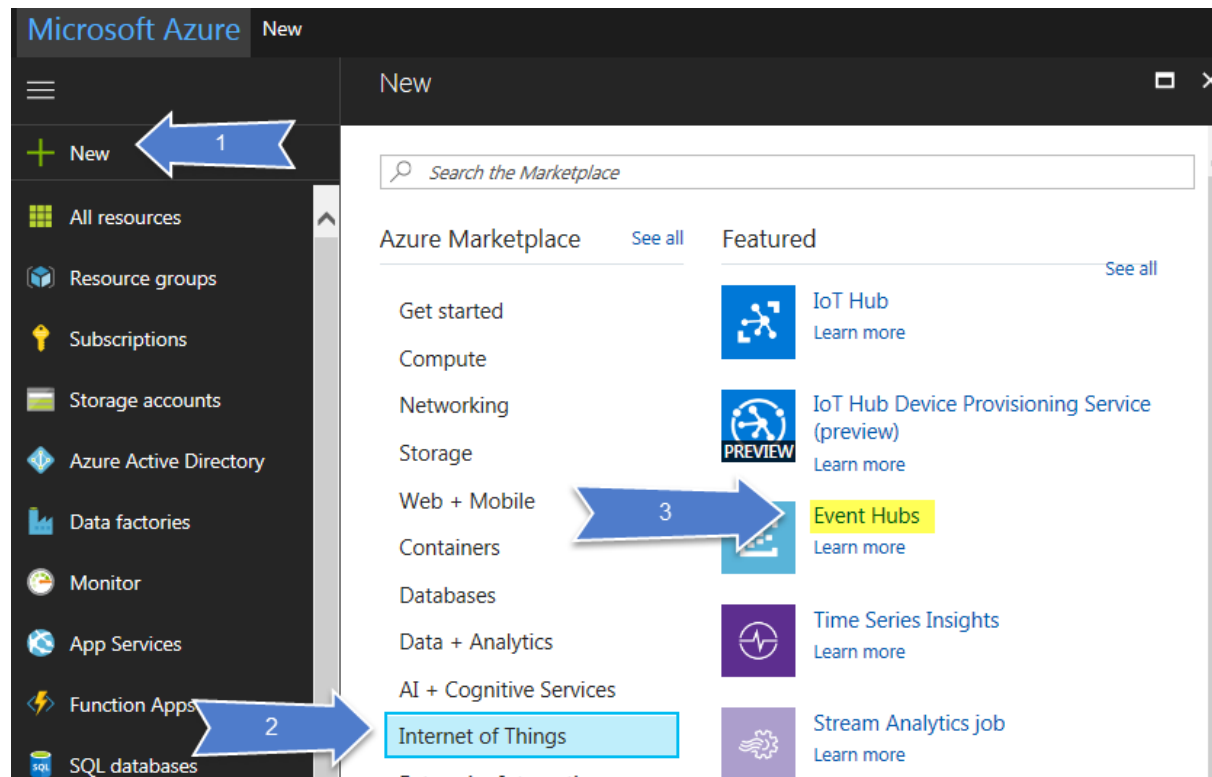
- Event Hubs .NET SDK
 - Based on .NET Standard 1.3
 - <https://github.com/Azure/azure-event-hubs-dotnet>
- Event Hubs Java SDK
 - <https://github.com/Azure/azure-event-hubs-java>

Not official release: (updated by *Microsoft employees and external contributors in their free time*)

- Azure Event Hub Client for Node.js

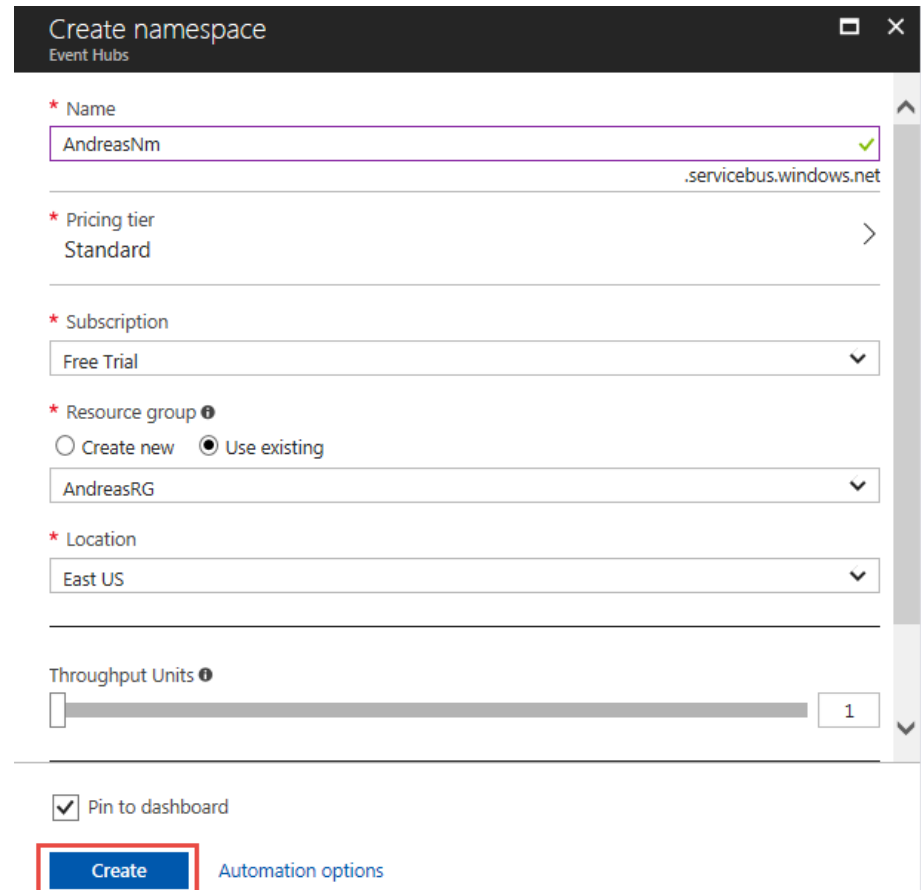
Create an Event Hub namespace

- Login to Azure and select New -> Internet of Things -> Event Hubs



Create an Event Hub namespace

- Create a unique name for your namespace
- Create a new RG (or use one that you already have)
- Select your location
- Select Pin to dashboard
- Select Create

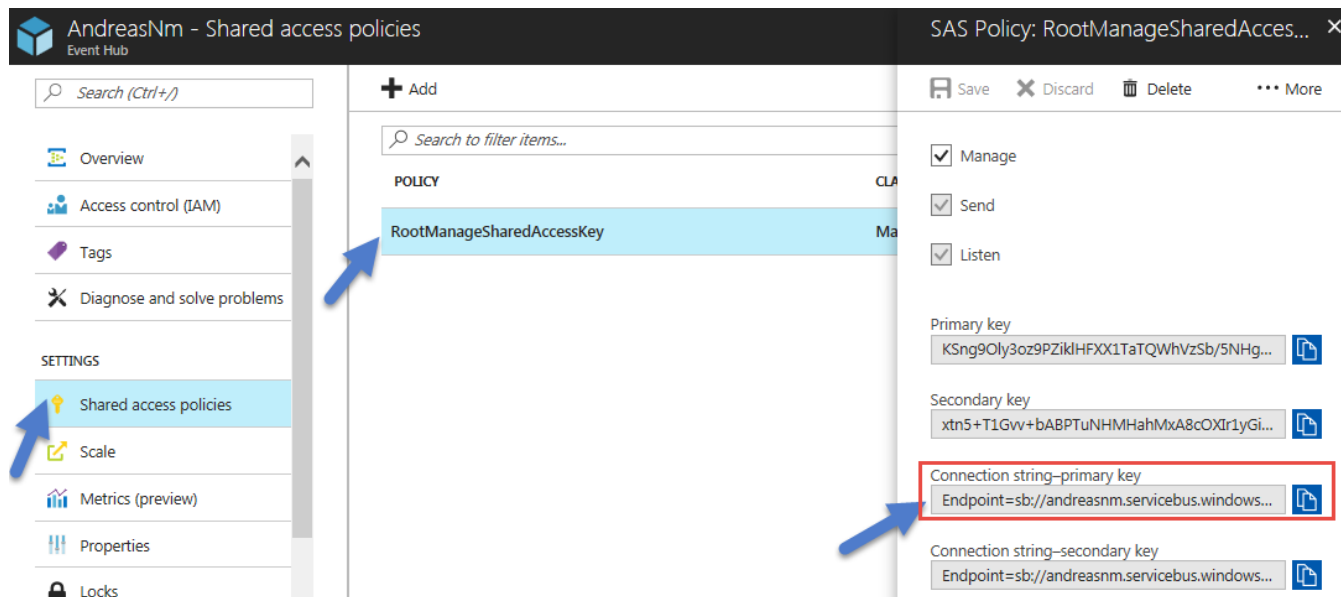


The screenshot shows the 'Create namespace' form for Event Hubs. The form is titled 'Create namespace' with a subtitle 'Event Hubs'. It contains several fields and options:

- Name:** A text input field containing 'AndreasNm' with a green checkmark on the right. Below it, the suffix '.servicebus.windows.net' is visible.
- Pricing tier:** A dropdown menu showing 'Standard' with a right arrow.
- Subscription:** A dropdown menu showing 'Free Trial' with a downward arrow.
- Resource group:** Radio buttons for 'Create new' and 'Use existing' (selected). Below is a dropdown menu showing 'AndreasRG' with a downward arrow.
- Location:** A dropdown menu showing 'East US' with a downward arrow.
- Throughput Units:** A slider bar with a value of '1' and a downward arrow.
- Pin to dashboard:** A checkbox that is checked.
- Create:** A blue button with a red border, highlighted by a red rectangle.
- Automation options:** A link text next to the 'Create' button.

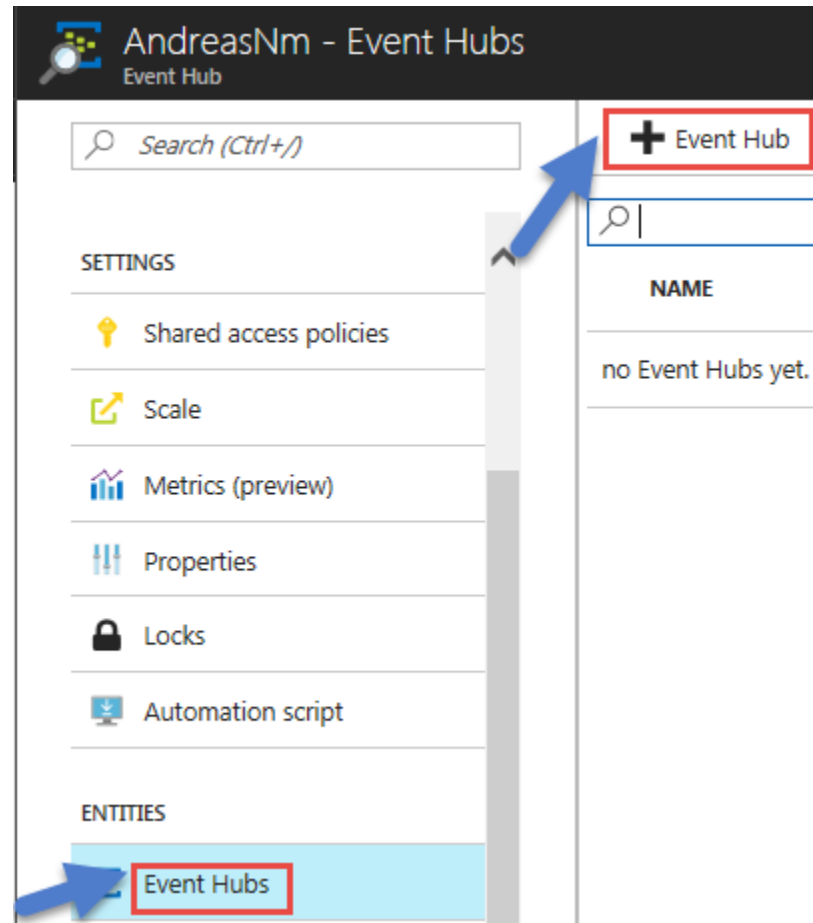
Find your Connection String

- Go to your newly created Event Hub and select Shared Access Policies under Settings
- Select the RootManageSharedAccessKey and then save your connection string (you will need this later on)



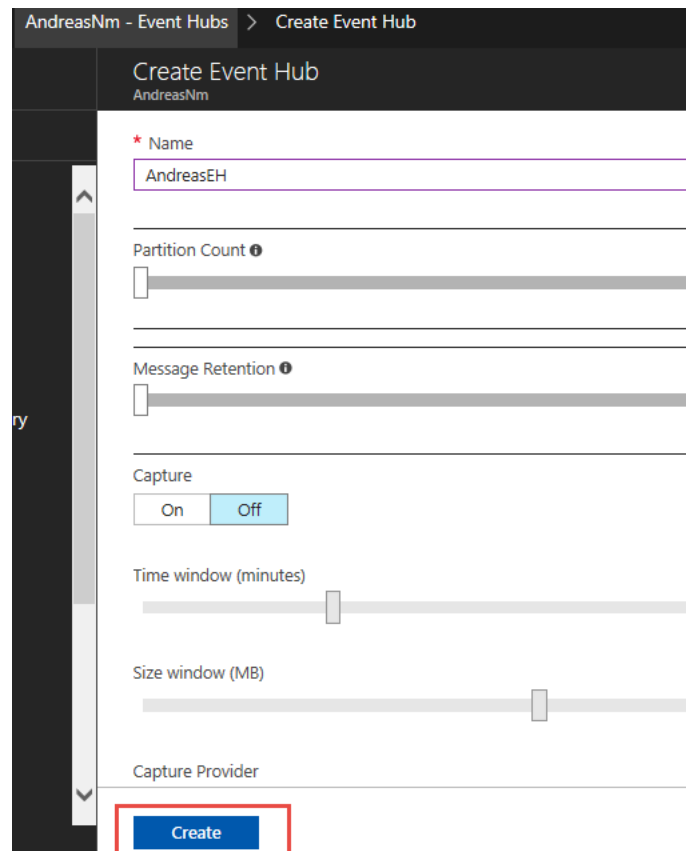
Create an Event Hub

- Click Event Hubs under Entities in your namespace
- Select + Event Hub



Create an Event Hub

- Add a name for your Event Hub and then select Create

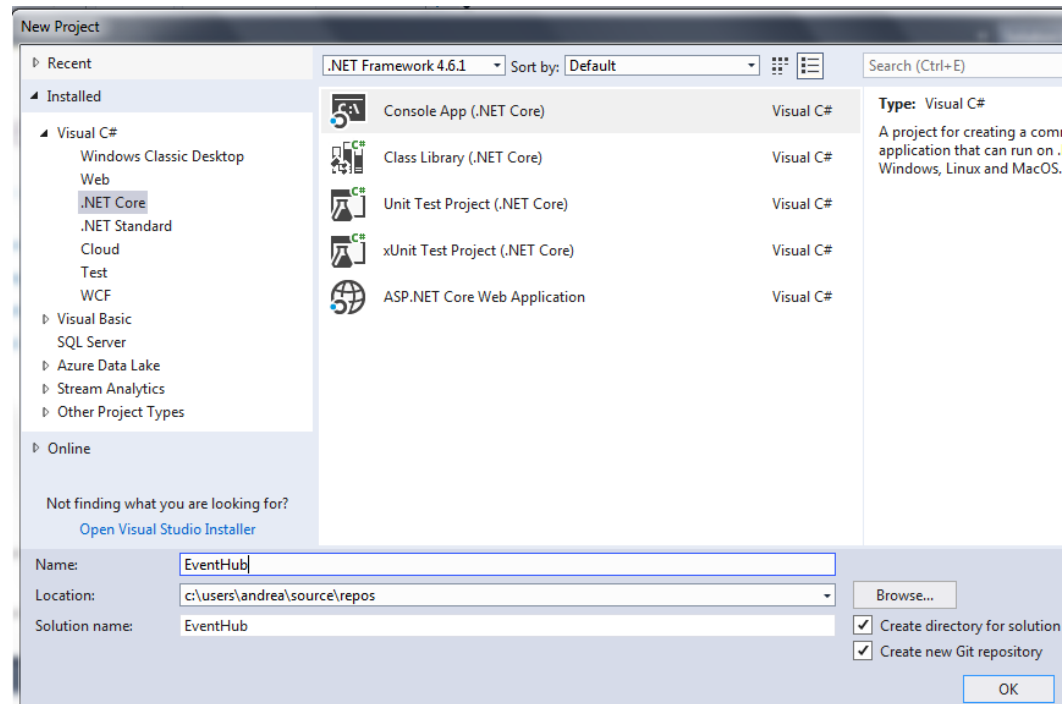


The screenshot shows the 'Create Event Hub' form in the Azure portal. The breadcrumb navigation at the top reads 'AndreasNm - Event Hubs > Create Event Hub'. The form title is 'Create Event Hub' with the subscription name 'AndreasNm' below it. The form contains the following fields and controls:

- Name:** A text input field with a red asterisk indicating it is required. The value 'AndreasEH' is entered.
- Partition Count:** A slider control with a help icon.
- Message Retention:** A slider control with a help icon.
- Capture:** A toggle switch with 'On' and 'Off' buttons. The 'Off' button is currently selected.
- Time window (minutes):** A slider control.
- Size window (MB):** A slider control.
- Capture Provider:** A dropdown menu.
- Create:** A blue button at the bottom of the form, highlighted with a red rectangular border.

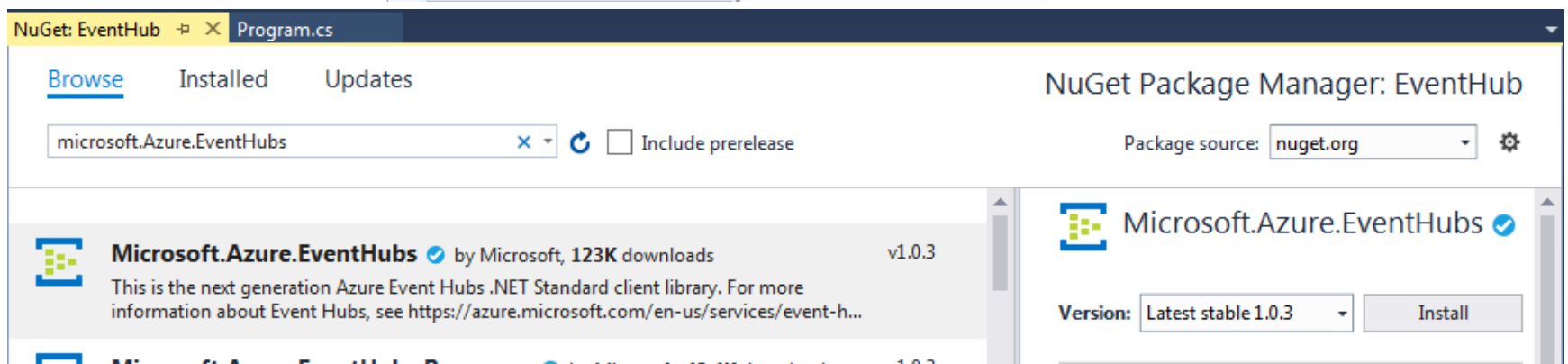
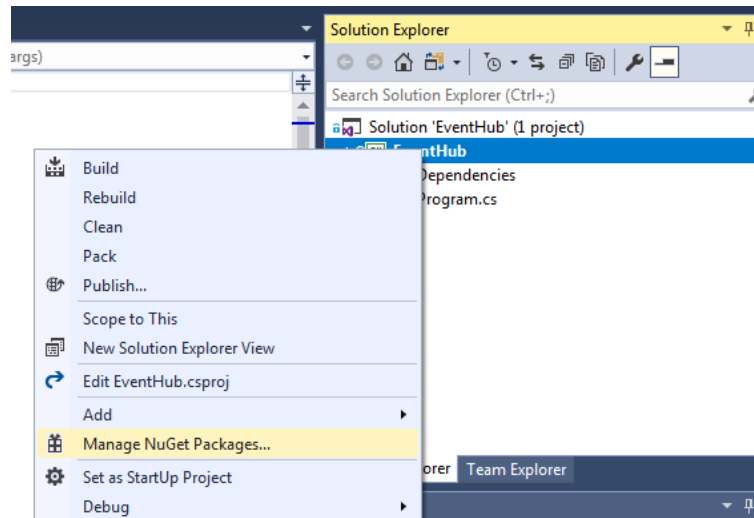
Create a Console App in VS

- Open VS and create a new .NET Console App by selecting File -> New -> Project
- Select the Console App under .NET Core, give it a name and then select OK



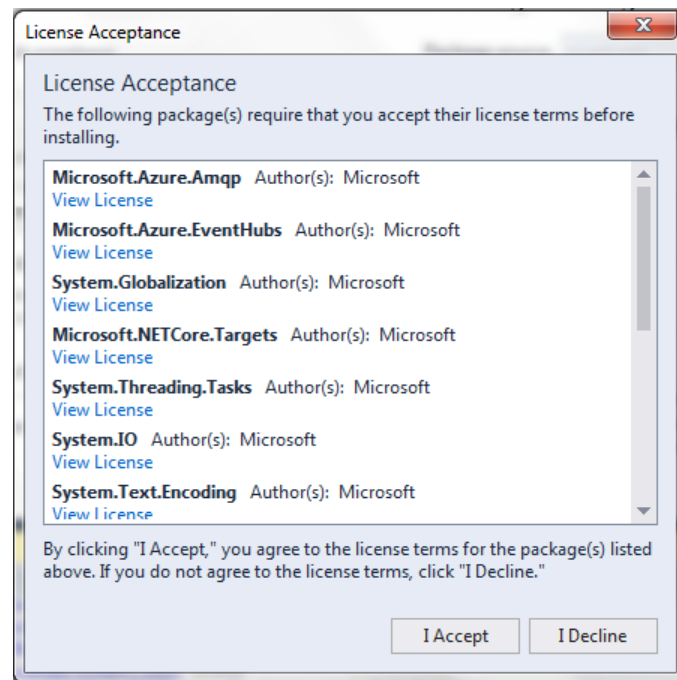
Add the Event Hubs NuGet package

- Right Click on your new project and select 'Manage NuGet Packages'
- Click Browse and search for Microsoft.Azure.EventHubs and then select to install it



Add the Event Hubs NuGet package

- You may be asked to preview the steps – select OK and then select I Accept for the installation of the License Acceptance



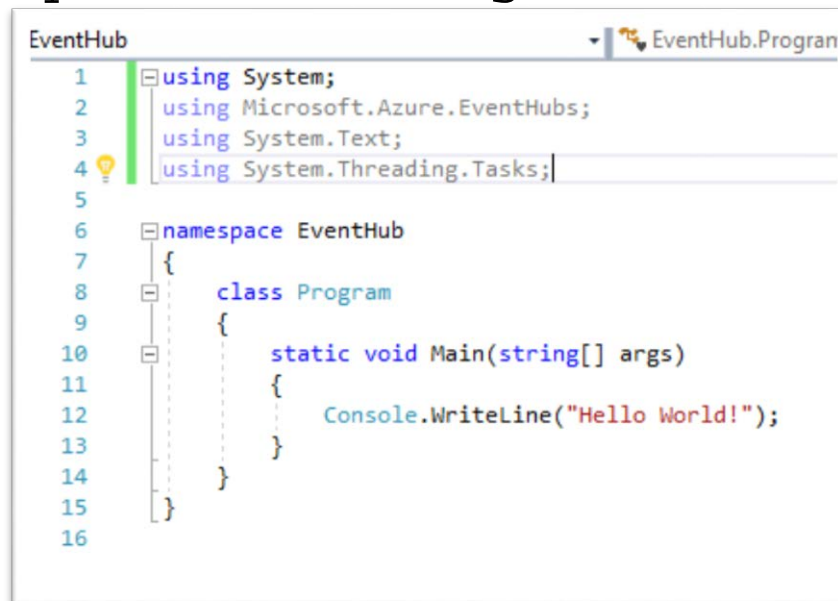
Sending Messages to your Event Hub

- Open your Program.cs file and add the three using statements to the top of the file:

```
using Microsoft.Azure.EventHubs;
```

```
using System.Text;
```

```
using System.Threading.Tasks;
```



Sending Messages to your Event Hub

- Add the following for your Event Hub. You will need to enter your connection string (slide 12) and entity path (the name of your Event hub) from Azure.

```
private static EventHubClient eventHubClient;  
private const string EhConnectionString = "{Event Hubs connection string}";  
private const string EhEntityPath = "{Event Hub path/name}";
```

Sending Messages to your Event Hub

```
namespace EventHub
{
    class Program
    {
        private static EventHubClient eventHubClient;
        private const string EhConnectionString = "{Event Hubs connection string}";
        private const string EhEntityPath = "{Event Hub path/name}";

        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Example without
the two strings
filled in

```
namespace EventHub
{
    class Program
    {
        private static EventHubClient eventHubClient;
        private const string EhConnectionString = "Endpoint=sb://andreasnm1.servicebus.windows.net/;SharedAccessKeyName=RootManag";
        private const string EhEntityPath = "andreaseh";
    }
}
```

Example with the
two strings filled in

Sending Messages to your Event Hub

- Add a new method to your Program Class and name it MainAsync

```
private static async Task MainAsync(string[] args)
{
    // Creates an EventHubsConnectionStringBuilder object from the connection string, and
    // sets the EntityPath. // Typically, the connection string should have the entity path in
    // it, but for the sake of this simple scenario // we are using the connection string from
    // the namespace.
    var connectionStringBuilder = new EventHubsConnectionStringBuilder(EhConnectionString)
    {
        EntityPath = EhEntityPath
    };
    eventHubClient =
    EventHubClient.CreateFromConnectionString(connectionStringBuilder.ToString());

    await SendMessagesToEventHub(100);

    await eventHubClient.CloseAsync();

    Console.WriteLine("Press ENTER to exit.");
    Console.ReadLine();
}
```

Sending Messages to your Event Hub

```
class Program
{
    private static EventHubClient eventHubClient;
    private const string EhConnectionString = "Endpoint=sb://andreasnm1.servicebus.windows.net/;SharedAccessKeyName=";
    private const string EhEntityPath = "AndreasNm1/andreaseh";

    private static async Task MainAsync(string[] args)
    {
        // Creates an EventHubsConnectionStringBuilder object from the connection string, and sets the EntityPath.
        // Typically, the connection string should have the entity path in it, but for the sake of this simple scene
        // we are using the connection string from the namespace.
        var connectionStringBuilder = new EventHubsConnectionStringBuilder(EhConnectionString)
        {
            EntityPath = EhEntityPath
        };

        eventHubClient = EventHubClient.CreateFromConnectionString(connectionStringBuilder.ToString());

        await SendMessagesToEventHub(100);

        await eventHubClient.CloseAsync();

        Console.WriteLine("Press ENTER to exit.");
        Console.ReadLine();
    }
}
```

Sending Messages to your Event Hub

- Add another method to your Program class called `SendMessageToEventHub`

```
// Creates an event hub client and sends 100 messages to the event hub.
private static async Task SendMessageToEventHub(int numMessagesToSend)
{
    for (var i = 0; i < numMessagesToSend; i++)
    {
        try
        {
            var message = $"Message {i}";
            Console.WriteLine($"Sending message: {message}");
            await eventHubClient.SendAsync(new EventData(Encoding.UTF8.GetBytes(message)));
        }
        catch (Exception exception)
        {
            Console.WriteLine($"{DateTime.Now} > Exception: {exception.Message}");
        }
        await Task.Delay(10);
    }
    Console.WriteLine($"{numMessagesToSend} messages sent.");
}
```

Sending Messages to your Event Hub

```
Program.cs  x
EventHub.Program  MainAsync(string[] args)

private static EventHubClient eventHubClient;
private const string EhConnectionString = "Endpoint=sb://andreasnm1.servicebus.windows.net/;SharedAccessKeyName=
private const string EhEntityPath = "AndreasNm1/andreseh";

// Creates an event hub client and sends 100 messages to the event hub.
private static async Task SendMessagesToEventHub(int numMessagesToSend)
{
    for (var i = 0; i < numMessagesToSend; i++)
    {
        try
        {
            var message = $"Message {i}";
            Console.WriteLine($"Sending message: {message}");
            await eventHubClient.SendAsync(new EventData(Encoding.UTF8.GetBytes(message)));
        }
        catch (Exception exception)
        {
            Console.WriteLine($"{DateTime.Now} > Exception: {exception.Message}");
        }

        await Task.Delay(10);
    }

    Console.WriteLine($"{numMessagesToSend} messages sent.");
}

private static async Task MainAsync(string[] args)
```


Sending Messages to your Event Hub

- Add the following code to the main of your Program Class

```
MainAsync(args).GetAwaiter().GetResult();
```

```
};

eventHubClient = EventHubClient.CreateFromConnectionString(connectionStringBuilder.ToString());

await SendMessagesToEventHub(100);

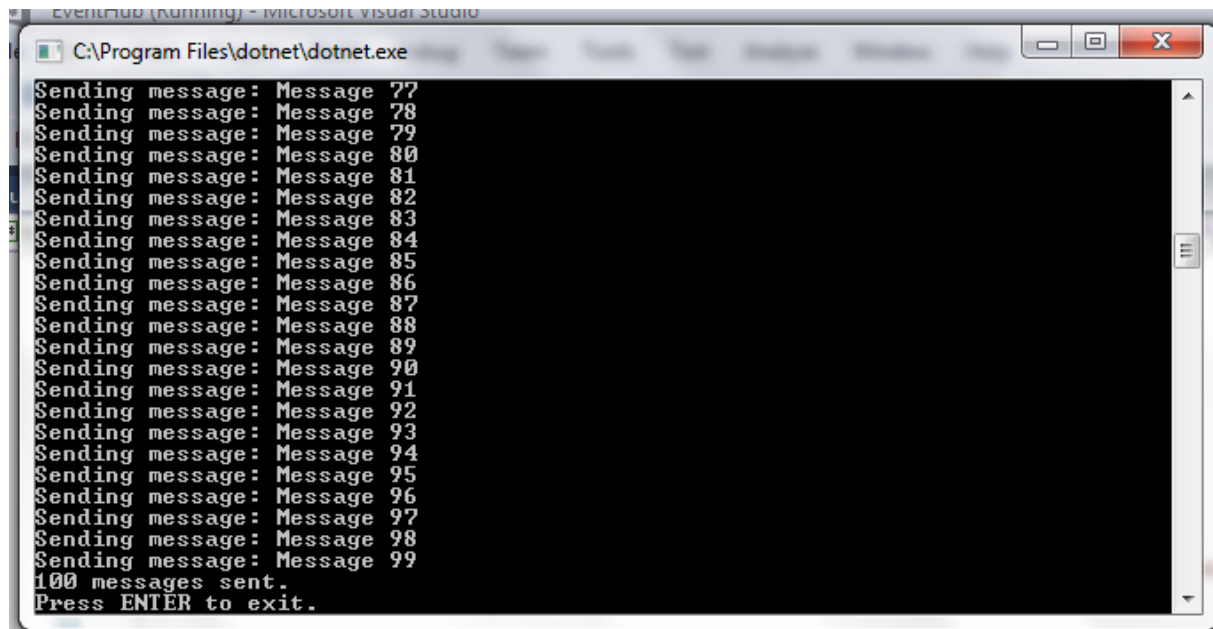
await eventHubClient.CloseAsync();

Console.WriteLine("Press ENTER to exit.");
Console.ReadLine();
}

static void Main(string[] args)
{
    MainAsync(args).GetAwaiter().GetResult();
    Console.WriteLine("Hello World!");
}
}
```

Run your program!

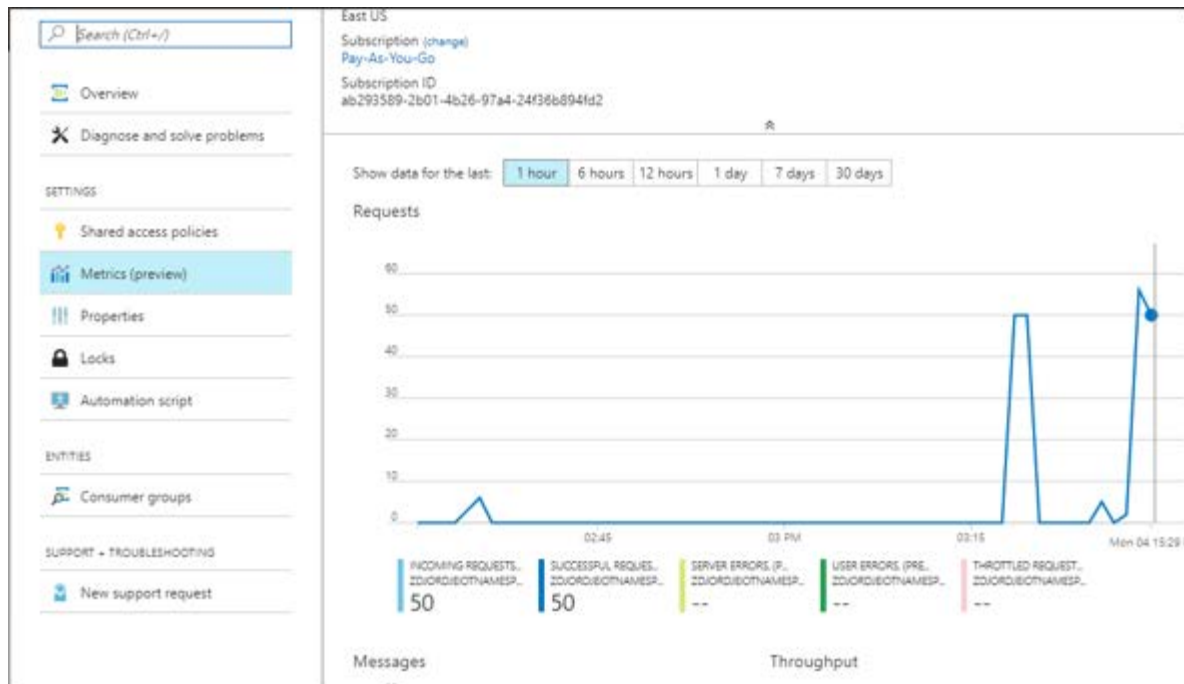
- Run the program to see your new messages that you sent to your Event Hub!
- Run the program by selecting your Solution and then select Debug and under Debug -> Start Without Debugging
- If VS complains that it cannot find `dotnet.exe`, add `C:\Program File\dotnet` (or wherever `dotnet.exe` resides) to your PATH variable.



```
EventHub (Running) - Microsoft Visual Studio
C:\Program Files\dotnet\dotnet.exe
Sending message: Message 77
Sending message: Message 78
Sending message: Message 79
Sending message: Message 80
Sending message: Message 81
Sending message: Message 82
Sending message: Message 83
Sending message: Message 84
Sending message: Message 85
Sending message: Message 86
Sending message: Message 87
Sending message: Message 88
Sending message: Message 89
Sending message: Message 90
Sending message: Message 91
Sending message: Message 92
Sending message: Message 93
Sending message: Message 94
Sending message: Message 95
Sending message: Message 96
Sending message: Message 97
Sending message: Message 98
Sending message: Message 99
100 messages sent.
Press ENTER to exit.
```

Where did Messages Go

- In Azure, under your Event Hub metrics you can see the program was run twice – just a few minutes apart!



III. IoT Hubs

IoT Hubs

- IoT Hubs are used for bi-directional communications
- IoT Hubs are used to send messages for different Internet of Things devices
- You can set up authentication for each of your connected devices to have confidentiality in your messaging



Source: <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-csharp-csharp-getstarted#introduction>

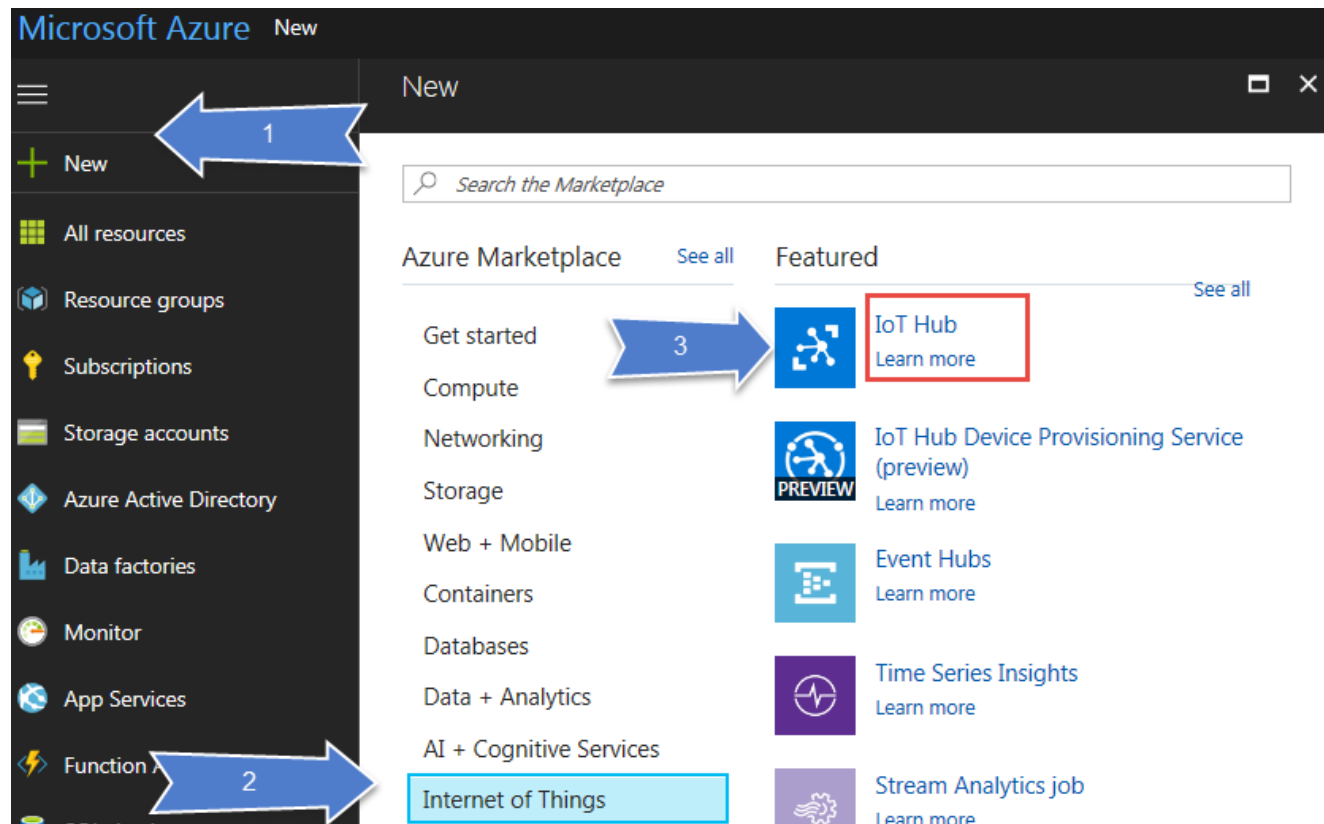
IoT Hubs SDKs

These are the 5 programming options available today for device and service IoT SDK's:

- IoT Hubs .NET SDK
 - <https://github.com/Azure/azure-iot-sdk-csharp/tree/master/device>
- IoT Java SDK
 - <https://github.com/Azure/azure-iot-sdk-java/tree/master/device>
- IoT Python SDK
 - <https://github.com/Azure/azure-iot-sdk-python/tree/master/device>
- IoT Node.JS SDK
 - <https://github.com/Azure/azure-iot-sdk-node/tree/master/device>
- IoT C SDK
 - <https://github.com/Azure/azure-iot-sdk-c>

Creating an IoT Hub

- In Azure, select New -> Internet of Things -> IoT Hub



Creating an IoT Hub

- Create a unique name
- Select or create your resource group
- Select a location
- Select Pin to dashboard
- Select Create

IoT hub
Microsoft

* Name
AndreasIoT ✓

* Pricing and scale tier
S1 - Standard >

* IoT Hub units ⓘ
1

* Device-to-cloud partitions ⓘ
4 partitions ▼

* Subscription
Free Trial ▼

* Resource group ⓘ
☐ Create new ☒ Use existing
AndreasRG ▼

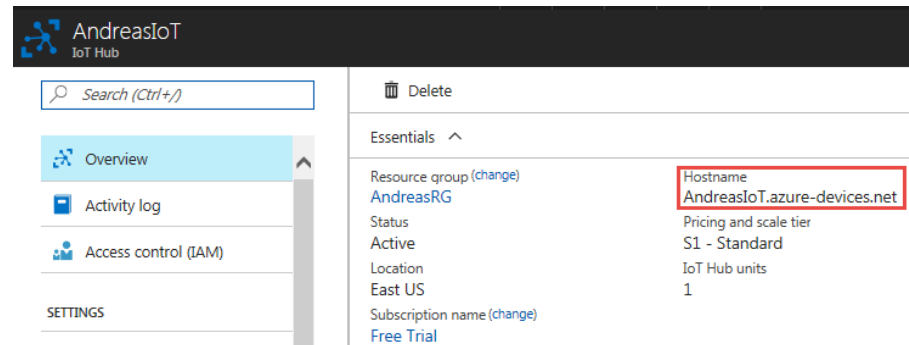
* Location
East US ▼

☒ Pin to dashboard

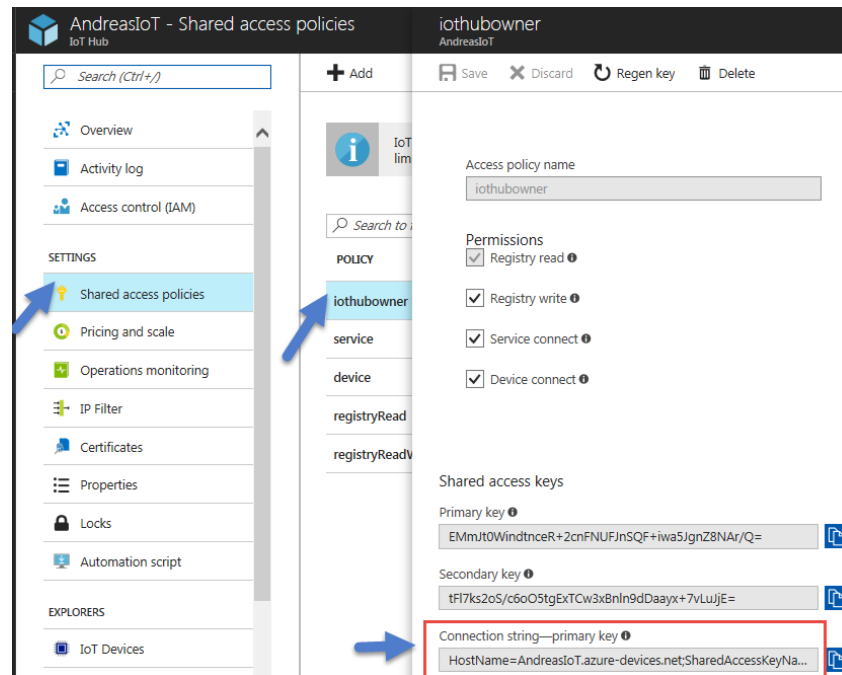
Create Automation options

Creating an IoT Hub

- In your IoT Hub, take note of your Hostname and store it in a notepad

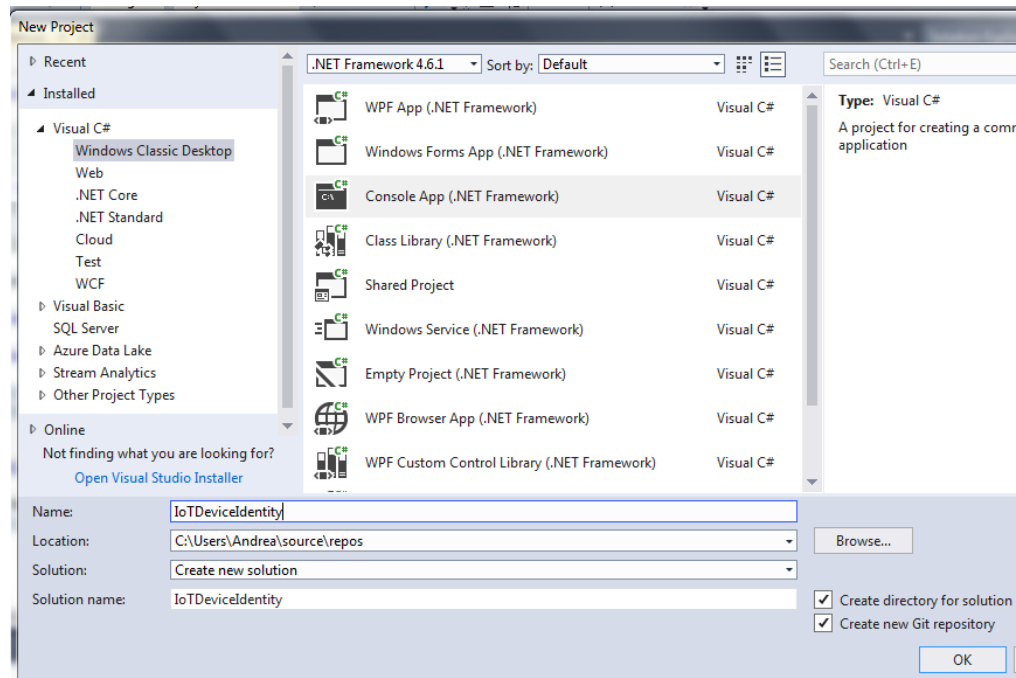


- Under your Shared access policies, select the iothubowner and copy your connection string, again storing it on a notepad



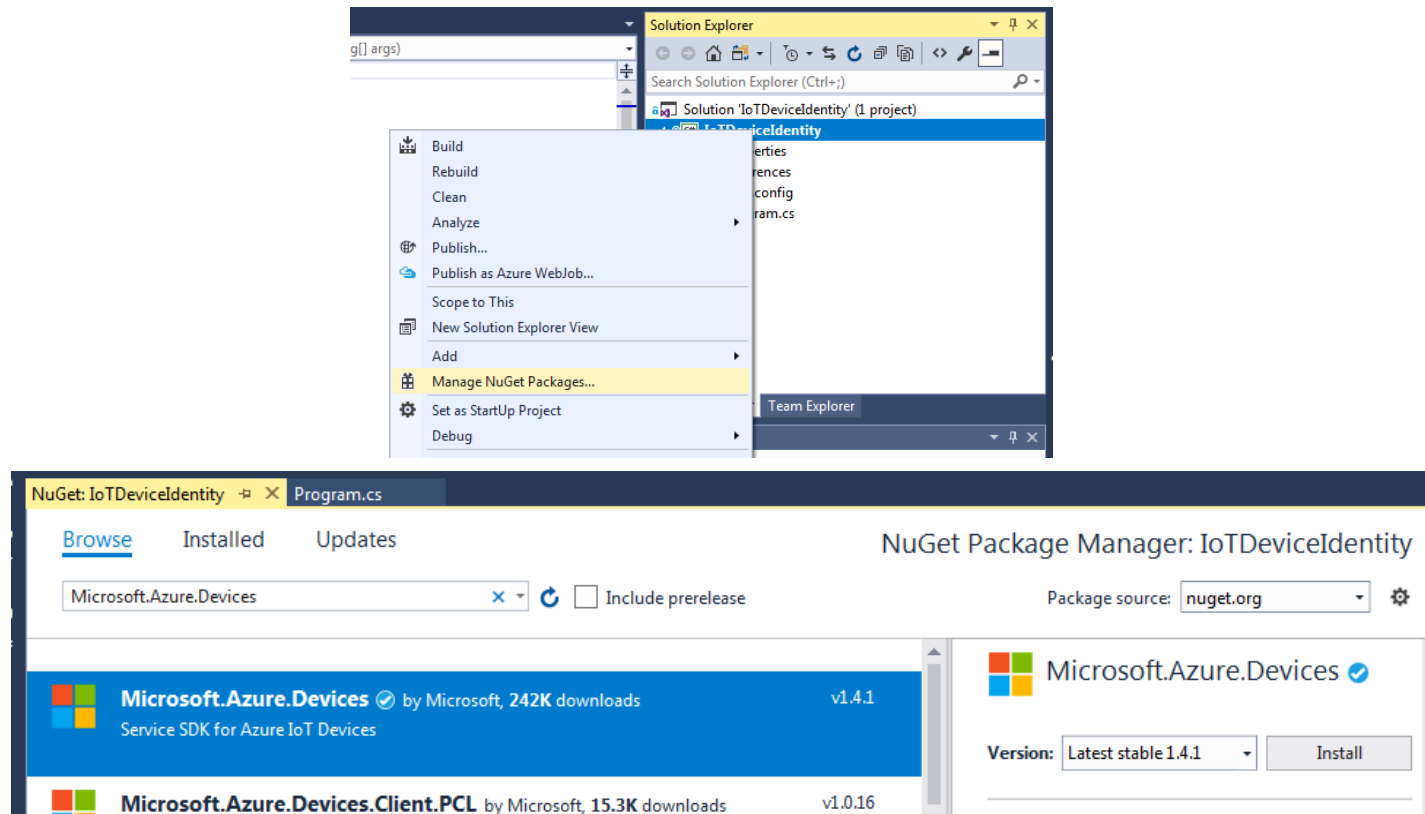
Creating an Identity

- In VS, Select File -> New -> Project
- Select the Windows Desktop section then Console App
- Add a Name and a Solution Name
- Note: You will work out of this solution for the entire IoT messaging



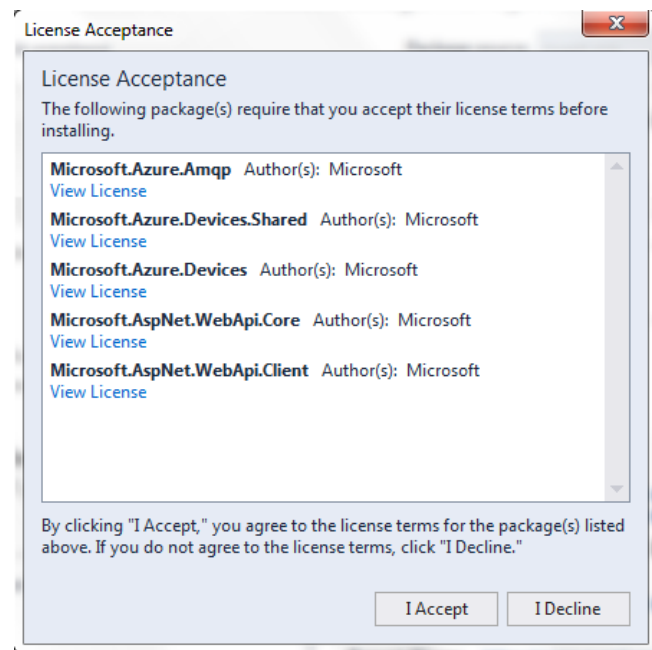
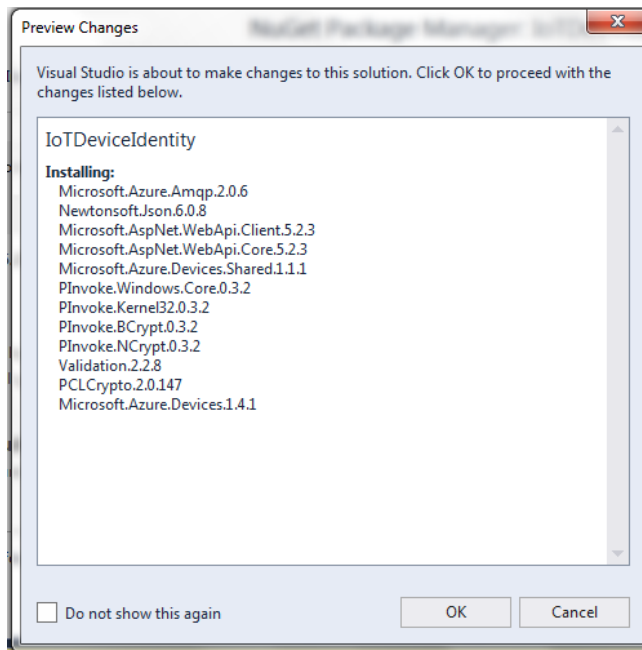
Creating an Identity

- Right click on your project and select Manage NuGet Packages
- Select Browse and search for **Microsoft.Azure.Devices** and select to Install it



Creating an Identity

- You will again need to select OK to preview your changes and Accept them

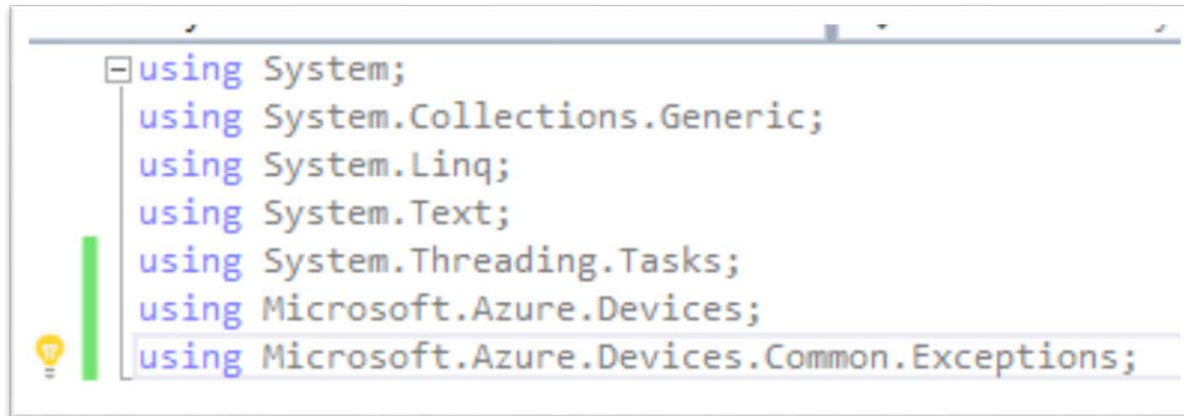


Creating an Identity

- Open your Program.cs file and add the two using statements:

```
using Microsoft.Azure.Devices;
```

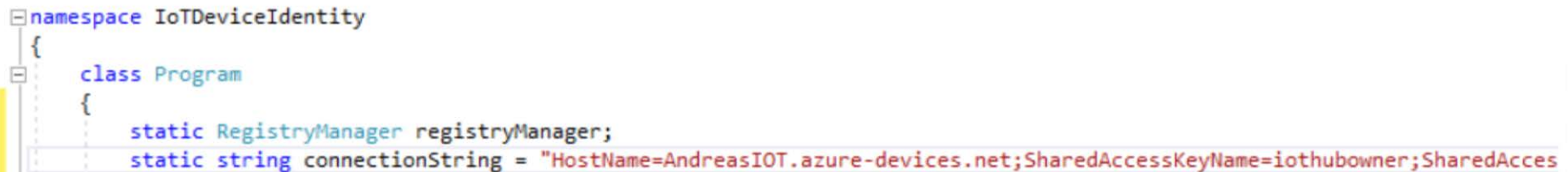
```
using Microsoft.Azure.Devices.Common.Exceptions;
```



Creating an Identity

- Add the following to your Program.CS file
You will need to enter your connection string (slide 33).

```
static RegistryManager registryManager;  
static string connectionString = "{iot hub connection  
string}";
```



```
namespace IoTDeviceIdentity  
{  
    class Program  
    {  
        static RegistryManager registryManager;  
        static string connectionString = "HostName=AndreasIoT.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=...";  
    }  
}
```

Creating an Identity

- Create a new method called `AddDeviceAsync()` in your `Program` class and add the following code

```
private static async Task AddDeviceAsync()  
{  
    string deviceId = "myFirstDevice";  
    Device device;  
    try  
    {  
        device = await registryManager.AddDeviceAsync(new Device(deviceId));  
    }  
    catch (DeviceAlreadyExistsException)  
    {  
        device = await registryManager.GetDeviceAsync(deviceId);  
    }  
    Console.WriteLine("Generated device key: {0}",  
        device.Authentication.SymmetricKey.PrimaryKey);  
}
```

Creating an Identity

```
private static async Task AddDeviceAsync()
{
    string deviceId = "myFirstDevice";
    Device device;
    try
    {
        device = await registryManager.AddDeviceAsync(new Device(deviceId));
    }
    catch (DeviceAlreadyExistsException)
    {
        device = await registryManager.GetDeviceAsync(deviceId);
    }
    Console.WriteLine("Generated device key: {0}", device.Authentication.SymmetricKey.PrimaryKey);
}
```


Creating an Identity

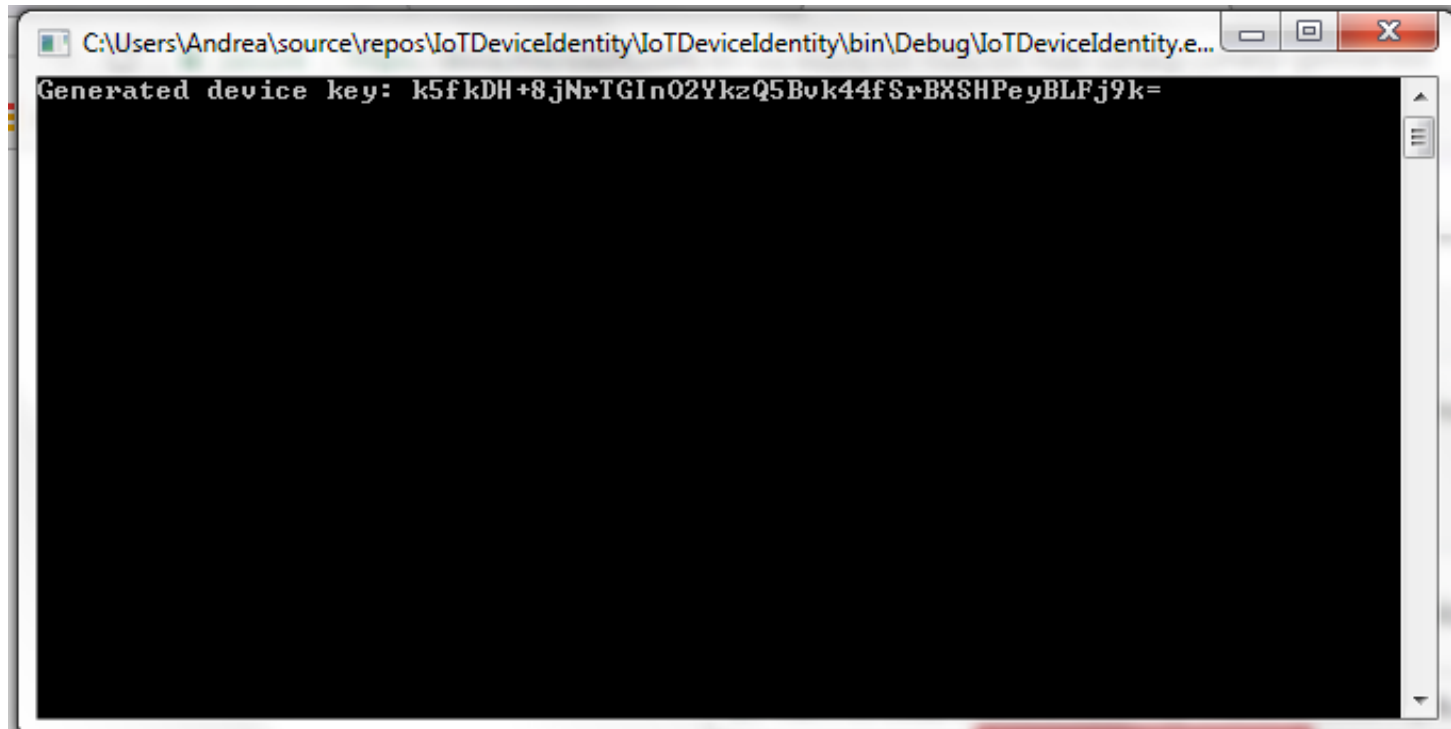
- Add the following in the main method

```
registryManager =  
RegistryManager.CreateFromConnectionString(connectionString);  
AddDeviceAsync().Wait(); Console.ReadLine();
```

```
static void Main(string[] args)  
{  
    registryManager = RegistryManager.CreateFromConnectionString(connectionString);  
    AddDeviceAsync().Wait();  
    Console.ReadLine();  
}
```

Run the application!

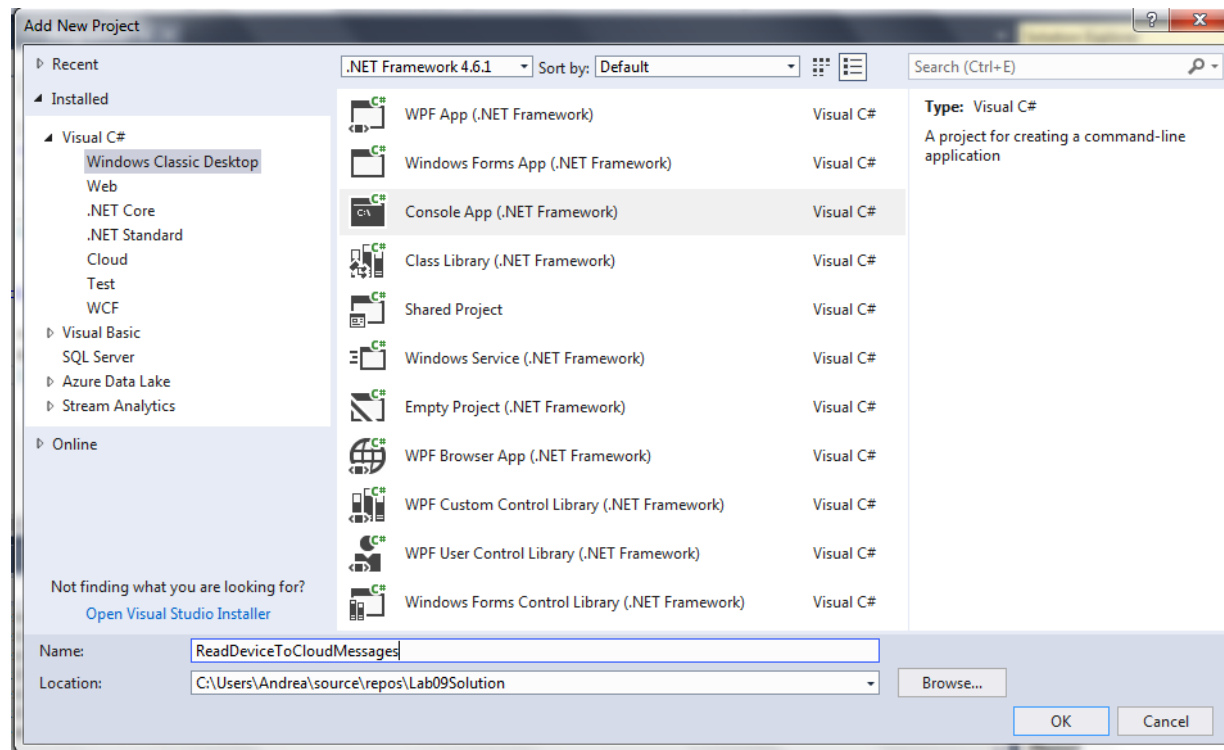
- Run this application and when the device key is given – **save it to your notepad**. As you will not be able to get it back.

A screenshot of a Windows command prompt window. The title bar shows the file path: C:\Users\Andrea\source\repos\IoTDeviceIdentity\IoTDeviceIdentity\bin\Debug\IoTDeviceIdentity.e... The window content displays the text: Generated device key: k5fkDH+8jNrTGI n02YkzQ5Bok44fSrBXSHPe yBLFj9k=. The text is in a monospaced font, and the background is black.

```
C:\Users\Andrea\source\repos\IoTDeviceIdentity\IoTDeviceIdentity\bin\Debug\IoTDeviceIdentity.e...  
Generated device key: k5fkDH+8jNrTGI n02YkzQ5Bok44fSrBXSHPe yBLFj9k=
```

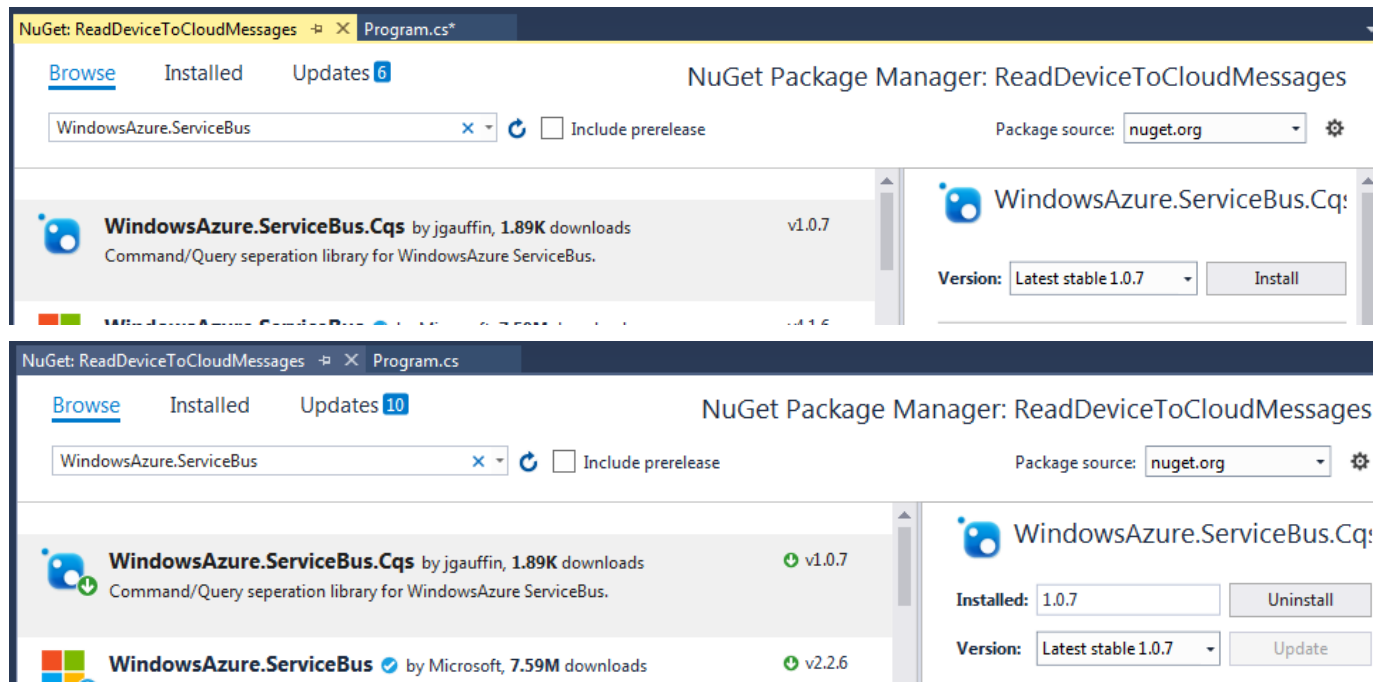
Working with device-to-cloud messages

- In VS, select File -> Add -> New Project
- Select Windows Classic Desktop -> Console App
- Add a name and select OK



Working with device-to-cloud messages

- Right click on your project and select Manage NuGet Packages
- Search for **WindowsAzure.ServiceBus** and select Install (Click ok and I accept to the pop ups)
- Again make sure to accept the changes

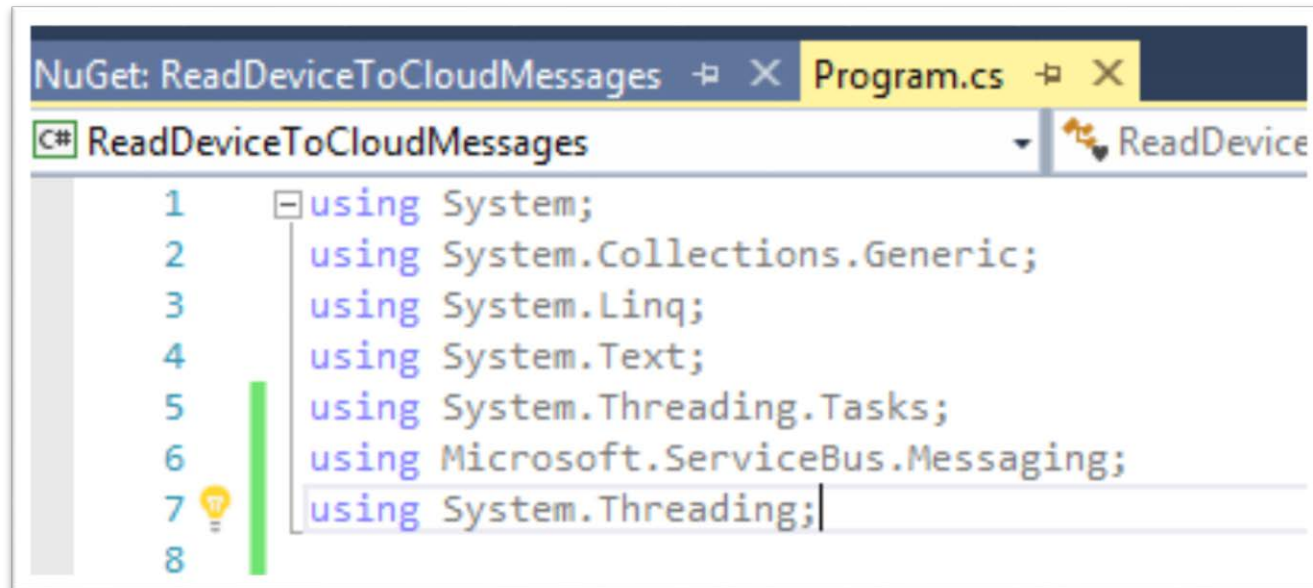


Working with device-to-cloud messages

- Open your Program.cs file and add the two using statements:

```
using Microsoft.ServiceBus.Messaging;
```

```
using System.Threading;
```

A screenshot of the Visual Studio IDE. The top of the window shows two tabs: 'NuGet: ReadDeviceToCloudMessages' and 'Program.cs'. The 'Program.cs' tab is active. Below the tabs, the file name 'C# ReadDeviceToCloudMessages' is displayed. The main area shows the code in Program.cs. Lines 1 through 8 are visible. Line 1: 'using System;'. Line 2: 'using System.Collections.Generic;'. Line 3: 'using System.Linq;'. Line 4: 'using System.Text;'. Line 5: 'using System.Threading.Tasks;'. Line 6: 'using Microsoft.ServiceBus.Messaging;'. Line 7: 'using System.Threading;'. Line 8: The cursor is at the end of line 7, after the semicolon. A green vertical bar is on the left side of the code editor, and a yellow lightbulb icon is next to line 7.

```
1 using System;  
2 using System.Collections.Generic;  
3 using System.Linq;  
4 using System.Text;  
5 using System.Threading.Tasks;  
6 using Microsoft.ServiceBus.Messaging;  
7 using System.Threading;  
8
```

Working with device-to-cloud messages

- Add the fields to the Program class. You will need to add your connection string here (slide 33).

```
static string connectionString = "{iothub connection string}";  
static string iotHubD2cEndpoint = "messages/events";  
static EventHubClient eventHubClient;
```



```
namespace ReadDeviceToCloudMessages  
{  
    static string connectionString = "HostName=AndreasIOT.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=";  
    static string iotHubD2cEndpoint = "messages/events";  
    static EventHubClient eventHubClient;
```

Working with device-to-cloud messages

- Add a `ReceiveMessagesFromDeviceAsync` method with the following code

```
private static async Task ReceiveMessagesFromDeviceAsync(string
partition, CancellationToken ct)
{
    var eventHubReceiver =
        eventHubClient.GetDefaultConsumerGroup().CreateReceiver(partition
        , DateTime.UtcNow);
    while (true)
    {
        if (ct.IsCancellationRequested) break;
        EventData eventData =
            await eventHubReceiver.ReceiveAsync();
        if (eventData == null) continue;
        string data = Encoding.UTF8.GetString(eventData.GetBytes());
        Console.WriteLine("Message received. Partition: {0} Data: '{1}'",
            partition, data);
    }
}
```

Working with device-to-cloud messages

```
class Program
{
    private static async Task ReceiveMessagesFromDeviceAsync(string partition, CancellationToken ct)
    {
        var eventHubReceiver = eventHubClient.GetDefaultConsumerGroup().CreateReceiver(partition, DateTime.UtcNow);
        while (true)
        {
            if (ct.IsCancellationRequested) break;
            EventData eventData = await eventHubReceiver.ReceiveAsync();
            if (eventData == null) continue;

            string data = Encoding.UTF8.GetString(eventData.GetBytes());
            Console.WriteLine("Message received. Partition: {0} Data: '{1}'", partition, data);
        }
    }
}
```


Working with device-to-cloud messages

- Add these lines to your main method:

```
Console.WriteLine("Receive messages. Ctrl-C to exit.\n");
eventHubClient =
EventHubClient.CreateFromConnectionString(connectionString,
IoTHubD2cEndpoint);
var d2cPartitions =
eventHubClient.GetRuntimeInformation().PartitionIds;
CancellationTokenSource cts = new CancellationTokenSource();
System.Console.CancelKeyPress += (s, e) =>
{
    e.Cancel = true;
    cts.Cancel();
    Console.WriteLine("Exiting...");
};
var tasks = new List<Task>();
foreach (string partition in d2cPartitions)
{
    tasks.Add(ReceiveMessagesFromDeviceAsync(partition, cts.Token));
}
Task.WaitAll(tasks.ToArray());
```

Working with device-to-cloud messages

```
static void Main(string[] args)
{
    Console.WriteLine("Receive messages. Ctrl-C to exit.\n");
    eventHubClient = EventHubClient.CreateFromConnectionString(connectionString, iotHubD2cEndpoint);

    var d2cPartitions = eventHubClient.GetRuntimeInformation().PartitionIds;

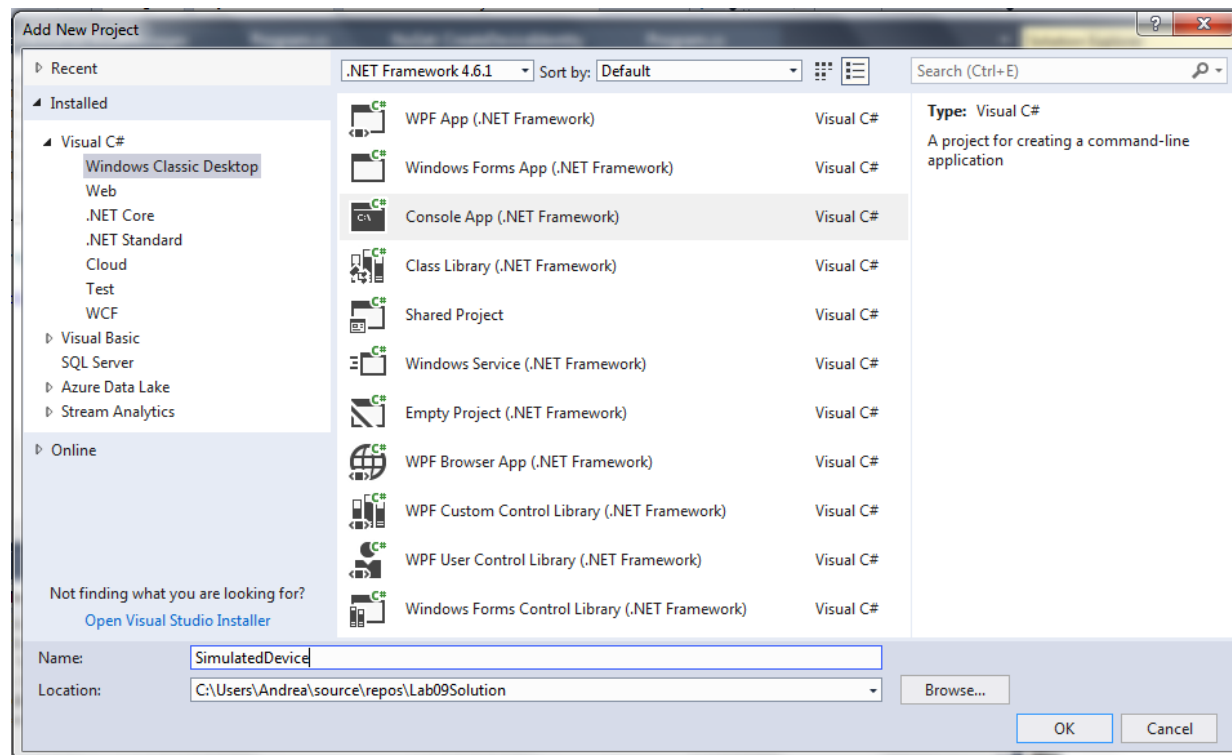
    CancellationTokensource cts = new CancellationTokensource();

    System.Console.CancelKeyPress += (s, e) =>
    {
        e.Cancel = true;
        cts.Cancel();
        Console.WriteLine("Exiting...");
    };

    var tasks = new List<Task>();
    foreach (string partition in d2cPartitions)
    {
        tasks.Add(ReceiveMessagesFromDeviceAsync(partition, cts.Token));
    }
    Task.WaitAll(tasks.ToArray());
}
```

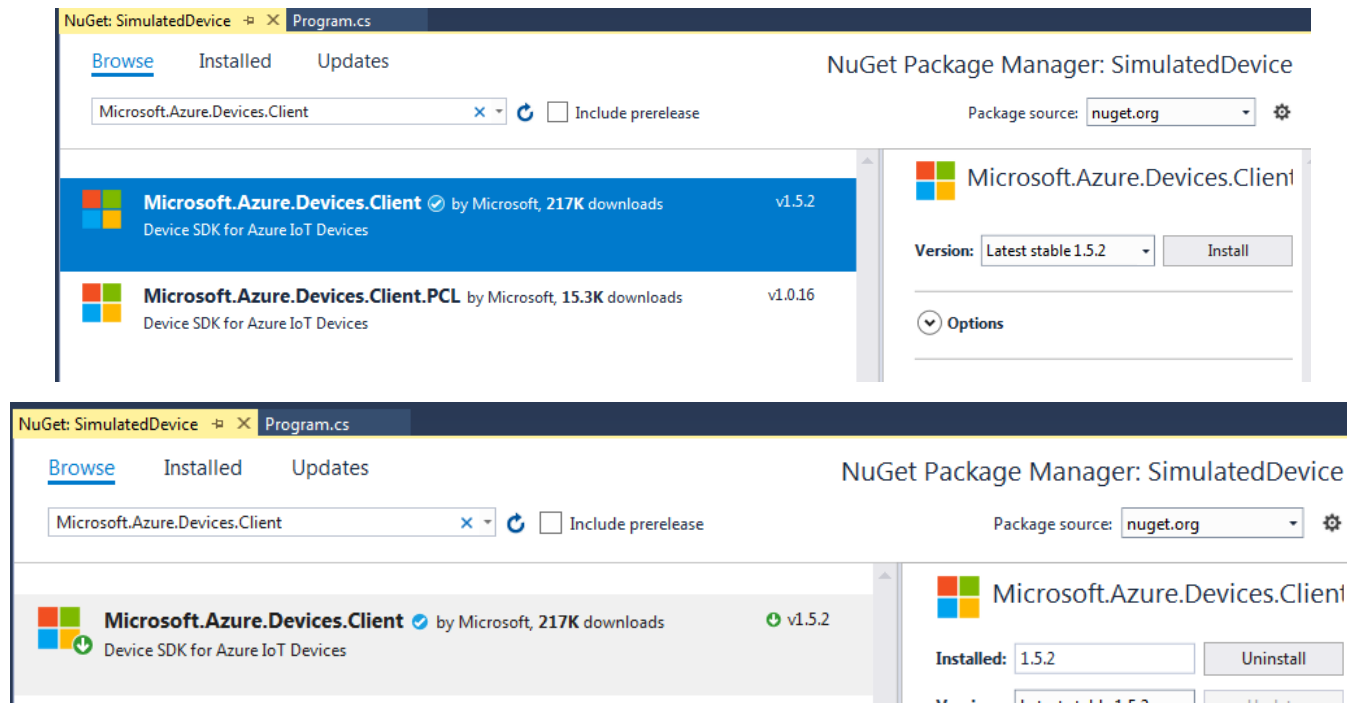
Create a device app

- In VS, select File -> Add -> New Project
- Select Windows Classic Desktop -> Console App
- Add a name and select OK



Create a device app

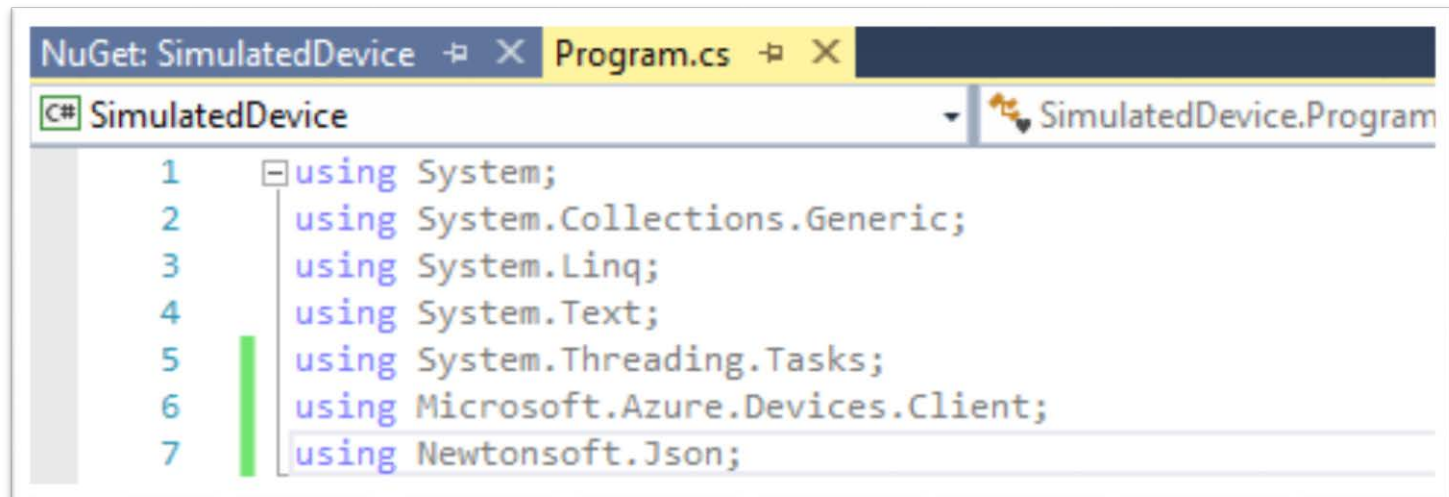
- Right click on your project and select Manage NuGet Packages
- Select Browse and search for **Microsoft.Azure.Devices.Client** and select to Install it
- Again, make sure to select ok and Accept the Install



Create a device app

- Open your Program.cs file and add the two using statements:

```
using Microsoft.Azure.Devices.Client;  
using Newtonsoft.Json;
```



Create a device app

- Add the fields to the Program class. You will need to add your IOT hub hostname (slide 33) and your device key (slide 42).

```
static DeviceClient deviceClient;  
static string iotHubUri = "{iot hub hostname}";  
static string deviceKey = "{device key}";
```



```
namespace SimulatedDevice  
{  
    class Program  
    {  
        static DeviceClient deviceClient;  
        static string iotHubUri = "AndreasIOT.azure-devices.net";  
        static string deviceKey = "k5fkDH+8jNrTGIn02YkzQ5Bvk44fSrBXSHPeYBLFj9k=";  
  
        static void Main(string[] args)  
        {  
        }  
    }  
}
```

Create a device app

- Add the following new method to your Program class

```
private static async void SendDeviceToCloudMessagesAsync()
{
    double minTemperature = 20;
    double minHumidity = 60;
    int messageId = 1;
    Random rand = new Random();

    while (true)
    {
        double currentTemperature = minTemperature + rand.NextDouble() * 15;
        double currentHumidity = minHumidity + rand.NextDouble() * 20;

        var telemetryDataPoint = new
        {
            messageId = messageId++,
            deviceId = "myFirstDevice",
            temperature = currentTemperature,
            humidity = currentHumidity
        };
        var messageString = JsonConvert.SerializeObject(telemetryDataPoint);
        var message = new Message(Encoding.ASCII.GetBytes(messageString));
        message.Properties.Add("temperatureAlert", (currentTemperature > 30) ? "true" :
"false");

        await deviceClient.SendEventAsync(message);
        Console.WriteLine("{0} > Sending message: {1}", DateTime.Now, messageString);

        await Task.Delay(1000);
    }
}
```

Create a device app

```
class Program
{
    static DeviceClient deviceClient;
    static string iotHubUri = "AndreasIOT.azure-devices.net";
    static string deviceKey = "k5fkDH+8jNrTGI02YkzQ5Bvk44fSrBXSHPeYBLFj9k=";

    private static async void SendDeviceToCloudMessagesAsync()
    {
        double minTemperature = 20;
        double minHumidity = 60;
        int messageId = 1;
        Random rand = new Random();

        while (true)
        {
            double currentTemperature = minTemperature + rand.NextDouble() * 15;
            double currentHumidity = minHumidity + rand.NextDouble() * 20;

            var telemetryDataPoint = new
            {
                messageId = messageId++,
            }
        }
    }
}
```


Create a device app

- The Send Device to Cloud Messages Async method that was just added will send a new message to the cloud every second.
- This method randomly generates a number to act like it is a humidity and a temperature sensor.

Create a device app

- Add the following to your Main method

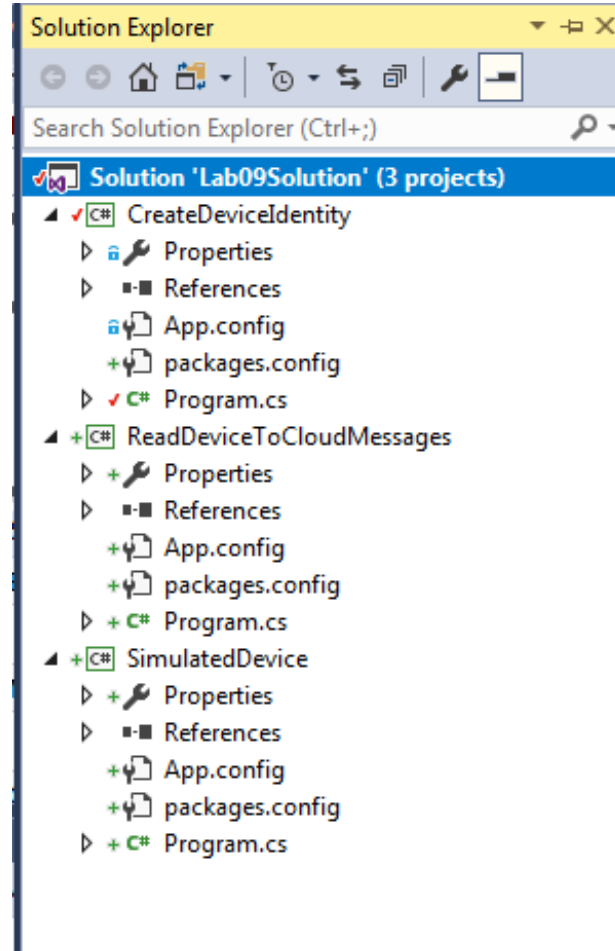
```
Console.WriteLine("Simulated device\n");  
deviceClient = DeviceClient.Create(iotHubUri, new  
DeviceAuthenticationWithRegistrySymmetricKey  
("myFirstDevice", deviceKey), TransportType.Mqtt);
```

```
SendDeviceToCloudMessagesAsync();  
Console.ReadLine();
```

```
static void Main(string[] args)  
{  
    Console.WriteLine("Simulated device\n");  
    deviceClient = DeviceClient.Create(iotHubUri, new DeviceAuthenticationWithRegistrySymmetricKey("myFirstDevice",  
    SendDeviceToCloudMessagesAsync();  
    Console.ReadLine();  
}
```

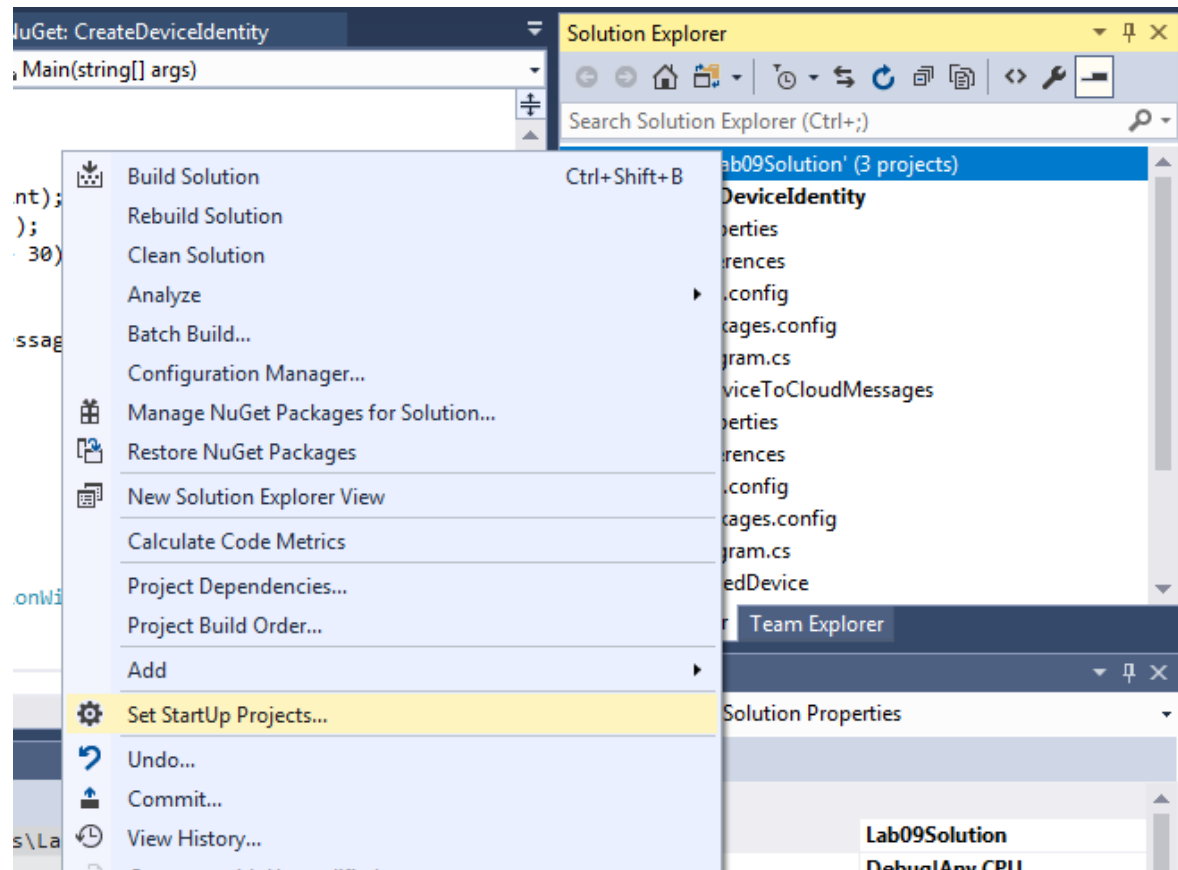
Your Solution

- In VS, your solution should be comprised of 3 projects and should look something like the following:



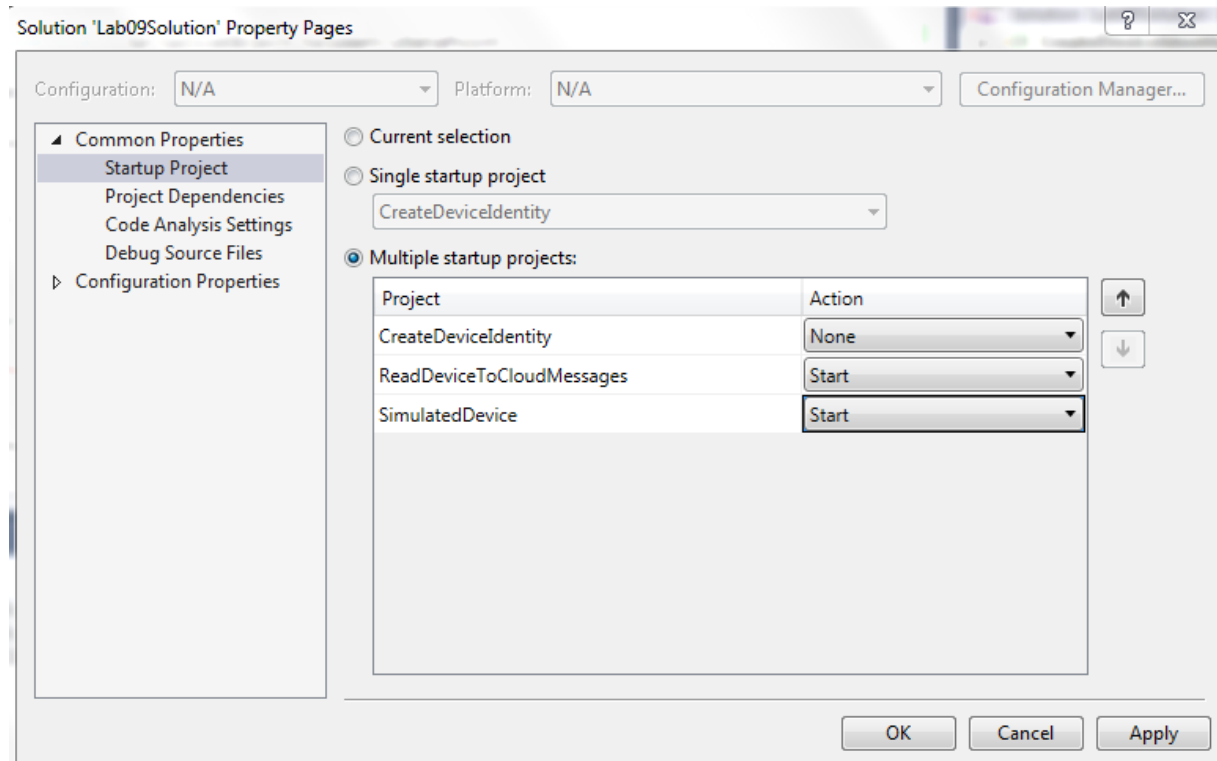
Running your app

- In VS, in the Solution Explorer right click on your **Solution** and select Set StartUp projects



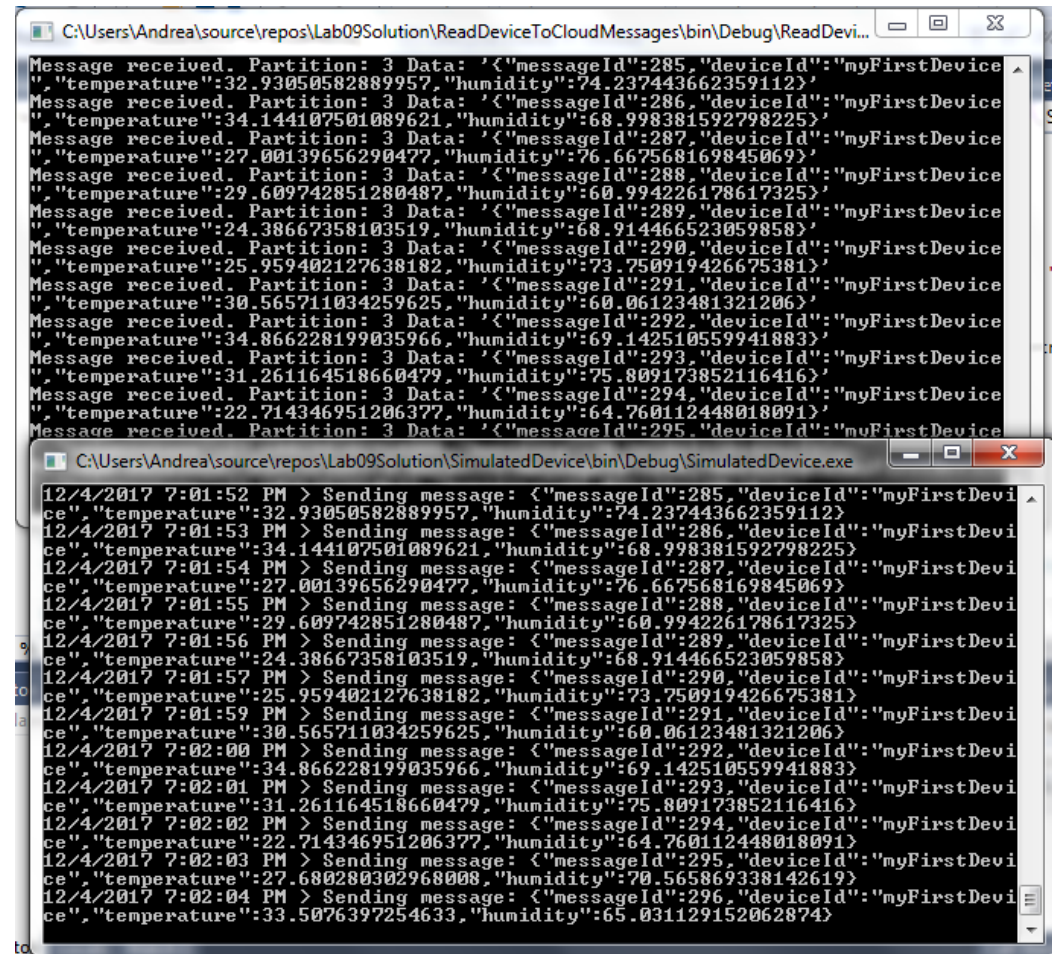
Running your app

- Select Multiple startup projects and under the action select 'Start' for both of the names of your projects from the create a device app section and the receive device to cloud messages section
- Select OK



Running your app

- Select F5 (or press start) to start both of the apps
- You will see two console outputs, one for each app
- The Read Device to Cloud app shows the message that your IoT hub is receiving
- The Simulated Device app will show your messages your app is sending to your IoT hub



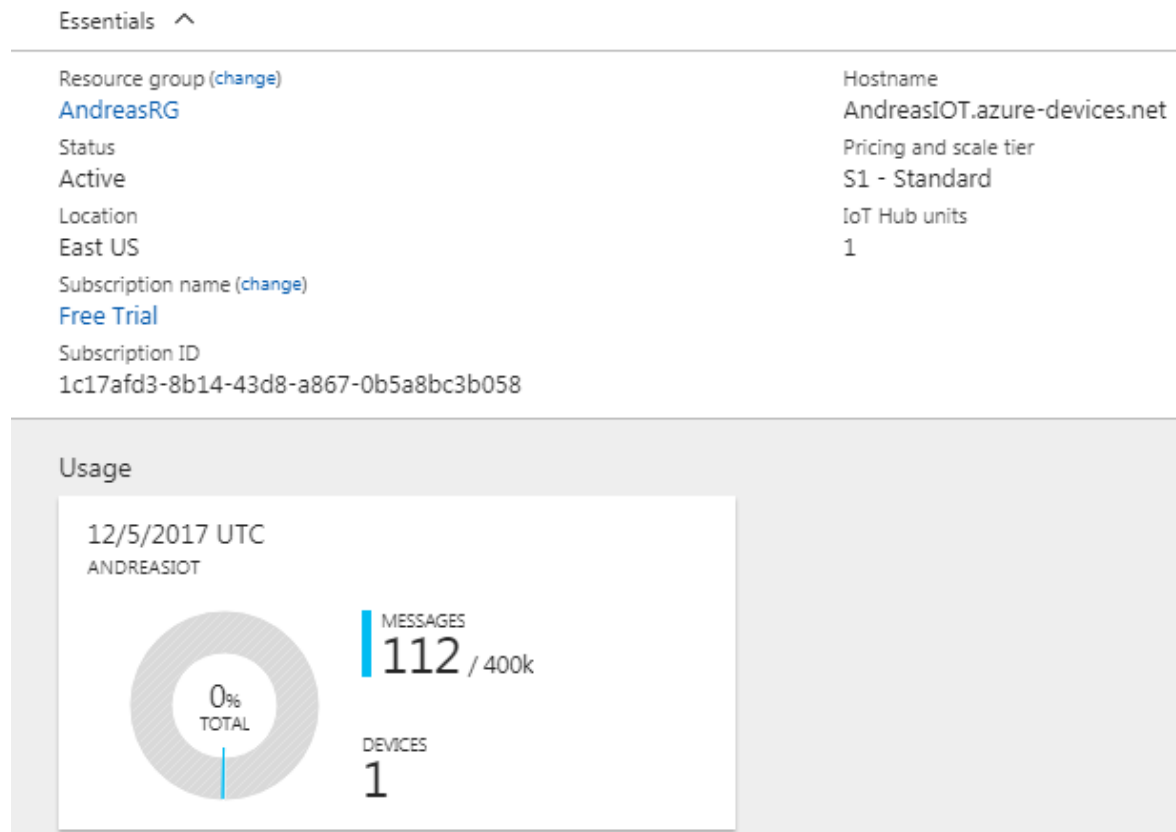
The image shows two overlapping console windows from a Visual Studio environment. The top window, titled 'C:\Users\Andrea\source\repos\Lab09Solution\ReadDeviceToCloudMessages\bin\Debug\ReadDev...', displays a series of messages received from a device. Each message is a JSON object containing 'messageId', 'deviceId', 'temperature', and 'humidity'. The bottom window, titled 'C:\Users\Andrea\source\repos\Lab09Solution\SimulatedDevice\bin\Debug\SimulatedDevice.exe', shows the corresponding messages being sent by the simulated device, including a timestamp and a prompt '> Sending message:'. Both windows show a sequence of 10 messages with increasing message IDs and varying temperature and humidity values.

```
Message received. Partition: 3 Data: '{"messageId":285,"deviceId":"myFirstDevice","temperature":32.93050582889957,"humidity":74.237443662359112}'
Message received. Partition: 3 Data: '{"messageId":286,"deviceId":"myFirstDevice","temperature":34.144107501089621,"humidity":68.998381592798225}'
Message received. Partition: 3 Data: '{"messageId":287,"deviceId":"myFirstDevice","temperature":27.00139656290477,"humidity":76.667568169845069}'
Message received. Partition: 3 Data: '{"messageId":288,"deviceId":"myFirstDevice","temperature":29.609742851280487,"humidity":60.994226178617325}'
Message received. Partition: 3 Data: '{"messageId":289,"deviceId":"myFirstDevice","temperature":24.38667358103519,"humidity":68.914466523059858}'
Message received. Partition: 3 Data: '{"messageId":290,"deviceId":"myFirstDevice","temperature":25.959402127638182,"humidity":73.750919426675381}'
Message received. Partition: 3 Data: '{"messageId":291,"deviceId":"myFirstDevice","temperature":30.565711034259625,"humidity":60.06123481321206}'
Message received. Partition: 3 Data: '{"messageId":292,"deviceId":"myFirstDevice","temperature":34.866228199035966,"humidity":69.142510559941883}'
Message received. Partition: 3 Data: '{"messageId":293,"deviceId":"myFirstDevice","temperature":31.261164518660479,"humidity":75.809173852116416}'
Message received. Partition: 3 Data: '{"messageId":294,"deviceId":"myFirstDevice","temperature":22.714346951206377,"humidity":64.760112448018091}'
Message received. Partition: 3 Data: '{"messageId":295,"deviceId":"myFirstDevice","temperature":27.680280302968008,"humidity":70.565869338142619}'

12/4/2017 7:01:52 PM > Sending message: '{"messageId":285,"deviceId":"myFirstDevice","temperature":32.93050582889957,"humidity":74.237443662359112}'
12/4/2017 7:01:53 PM > Sending message: '{"messageId":286,"deviceId":"myFirstDevice","temperature":34.144107501089621,"humidity":68.998381592798225}'
12/4/2017 7:01:54 PM > Sending message: '{"messageId":287,"deviceId":"myFirstDevice","temperature":27.00139656290477,"humidity":76.667568169845069}'
12/4/2017 7:01:55 PM > Sending message: '{"messageId":288,"deviceId":"myFirstDevice","temperature":29.609742851280487,"humidity":60.994226178617325}'
12/4/2017 7:01:56 PM > Sending message: '{"messageId":289,"deviceId":"myFirstDevice","temperature":24.38667358103519,"humidity":68.914466523059858}'
12/4/2017 7:01:57 PM > Sending message: '{"messageId":290,"deviceId":"myFirstDevice","temperature":25.959402127638182,"humidity":73.750919426675381}'
12/4/2017 7:01:59 PM > Sending message: '{"messageId":291,"deviceId":"myFirstDevice","temperature":30.565711034259625,"humidity":60.06123481321206}'
12/4/2017 7:02:00 PM > Sending message: '{"messageId":292,"deviceId":"myFirstDevice","temperature":34.866228199035966,"humidity":69.142510559941883}'
12/4/2017 7:02:01 PM > Sending message: '{"messageId":293,"deviceId":"myFirstDevice","temperature":31.261164518660479,"humidity":75.809173852116416}'
12/4/2017 7:02:02 PM > Sending message: '{"messageId":294,"deviceId":"myFirstDevice","temperature":22.714346951206377,"humidity":64.760112448018091}'
12/4/2017 7:02:03 PM > Sending message: '{"messageId":295,"deviceId":"myFirstDevice","temperature":27.680280302968008,"humidity":70.565869338142619}'
12/4/2017 7:02:04 PM > Sending message: '{"messageId":296,"deviceId":"myFirstDevice","temperature":33.5076397254633,"humidity":65.031129152062874}'
```

Running your app

- In Azure, go to your IoT Hub and view the Usage under Overview
- This will show the number of messages you have sent your IoT hub



Summary

- We saw how to create an Event Hub and IoT Hub
- We created one .NET console app for your Event Hub and sent messages to your Event Hub
- We created three .NET console apps with IoT Hubs to send and receive messages to and from your IoT Hub