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

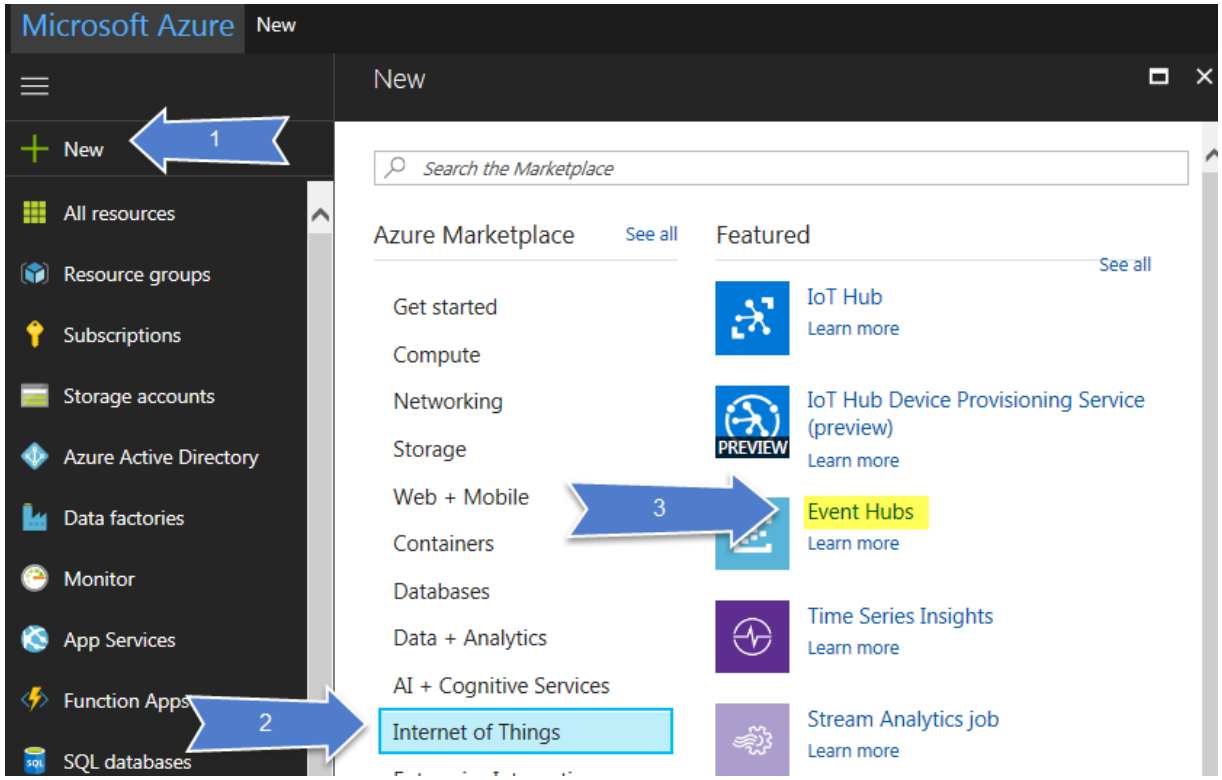
- Event Hubs Java SDK
  - https://github.com/Azure/azure-event-hubs-java

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

- Azure Event Hub Client for Node.js

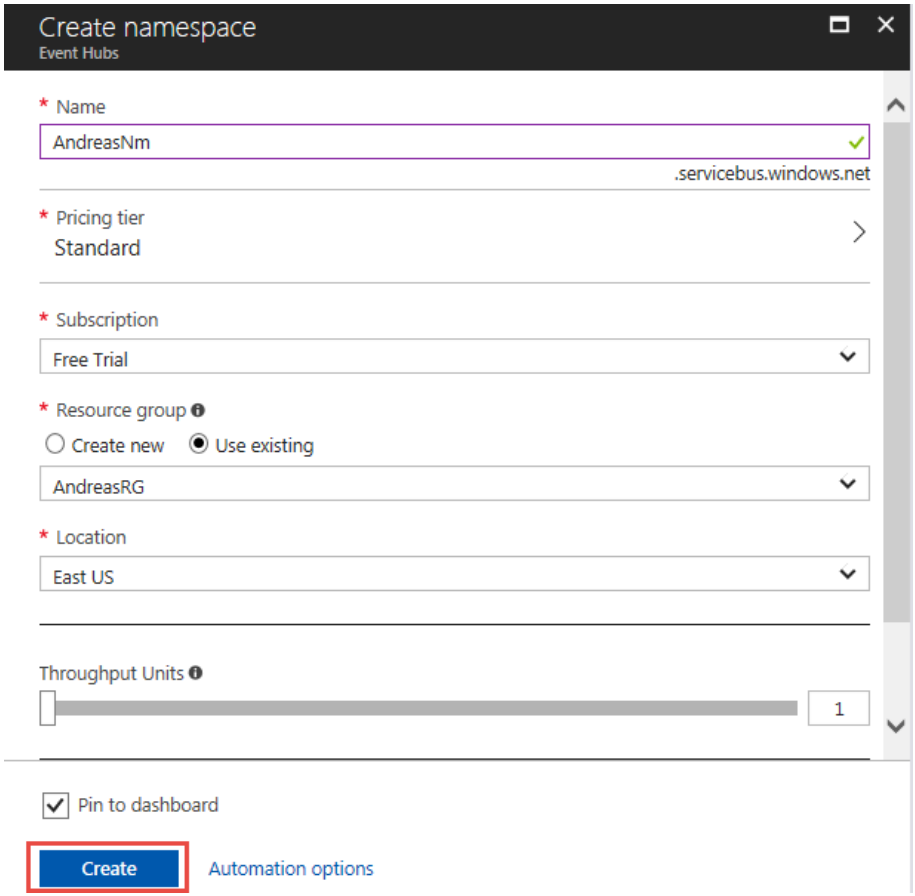
Create an Event Hub namespace

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



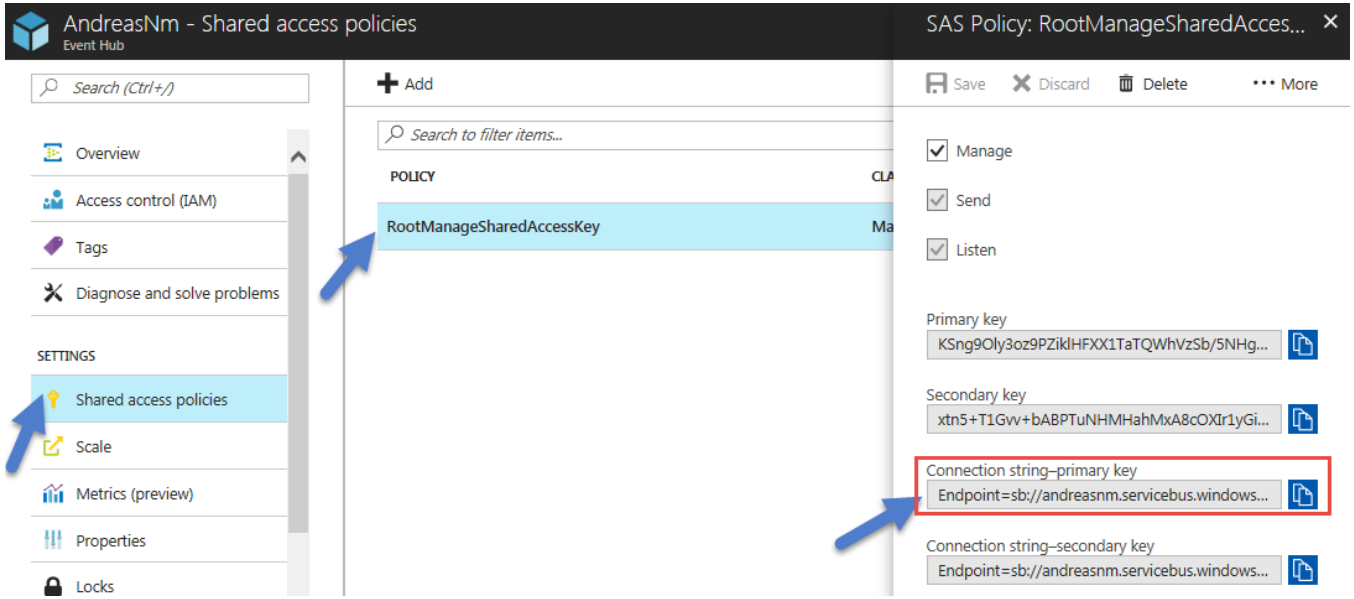
Create an Event Hub namespace

- 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



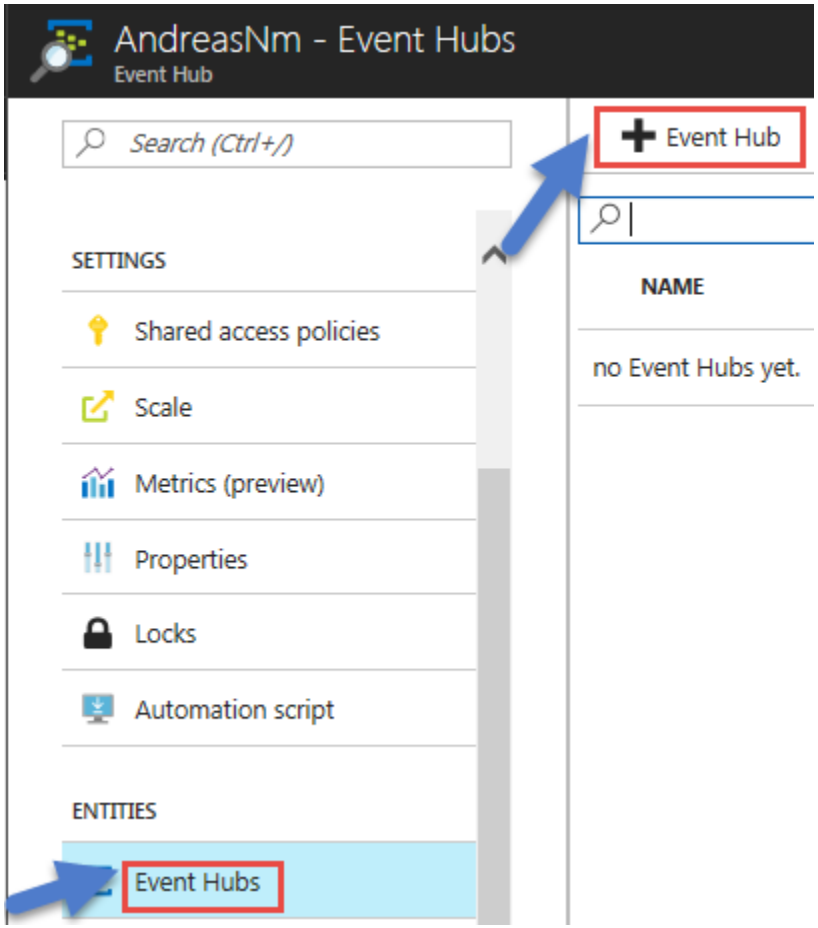
Find your Connection String

- 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)



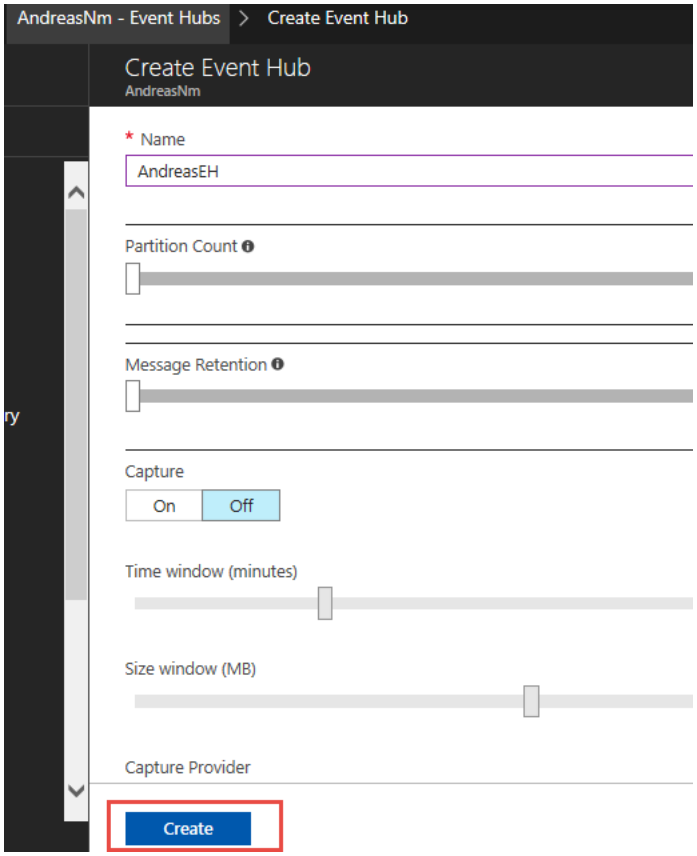
Create an Event Hub

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



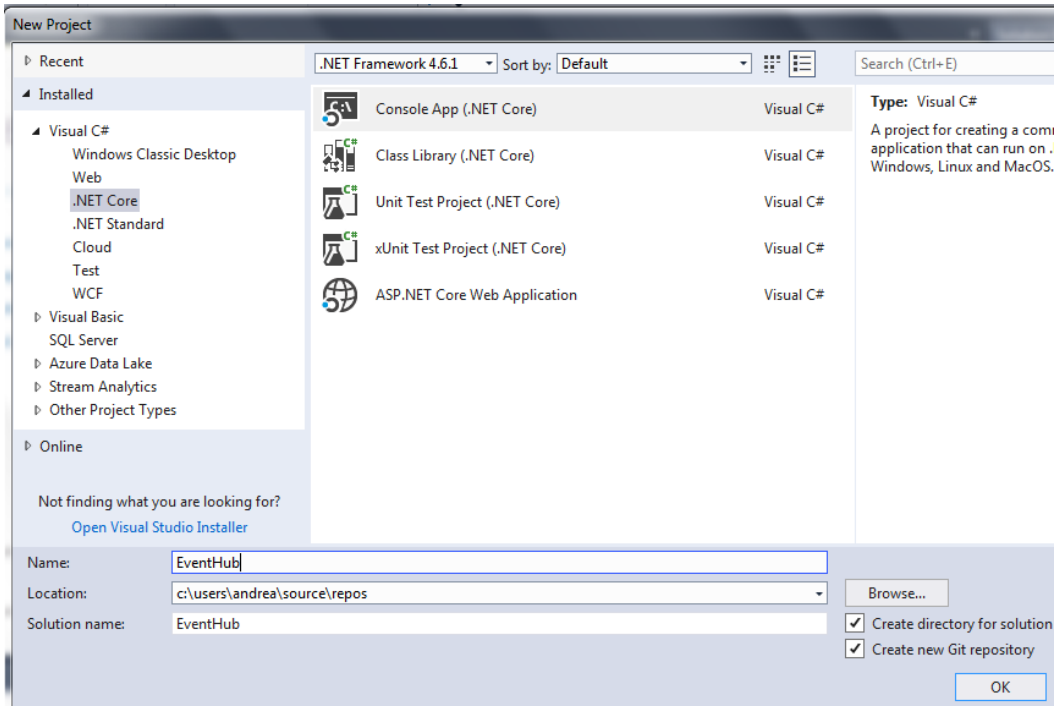
Create an Event Hub

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



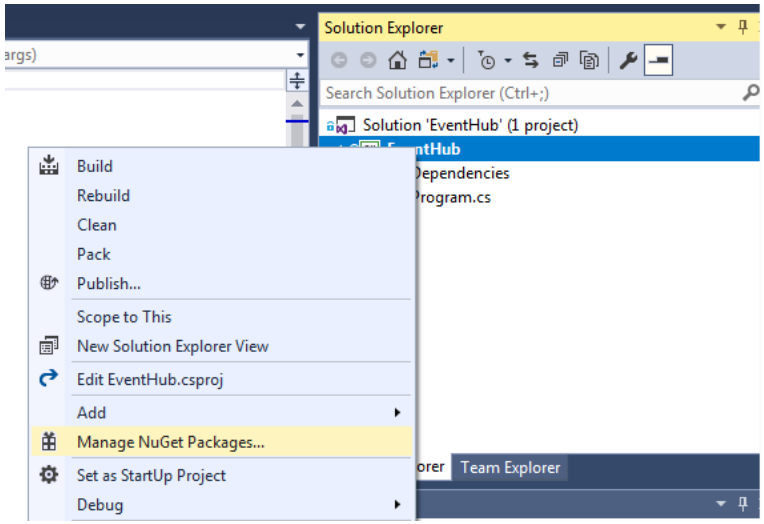
Create a Console App in VS

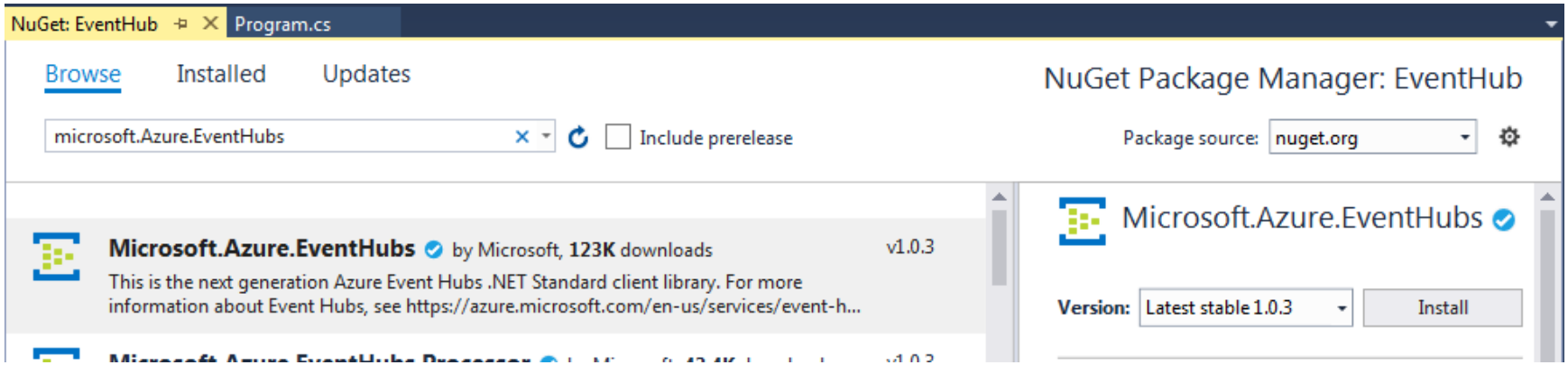
- Open VS and create a new .NET Console App by selecting File -> New -> Project
- Select the Console App under .NET Core, give it a name and then select OK



Add the Event Hubs NuGet package

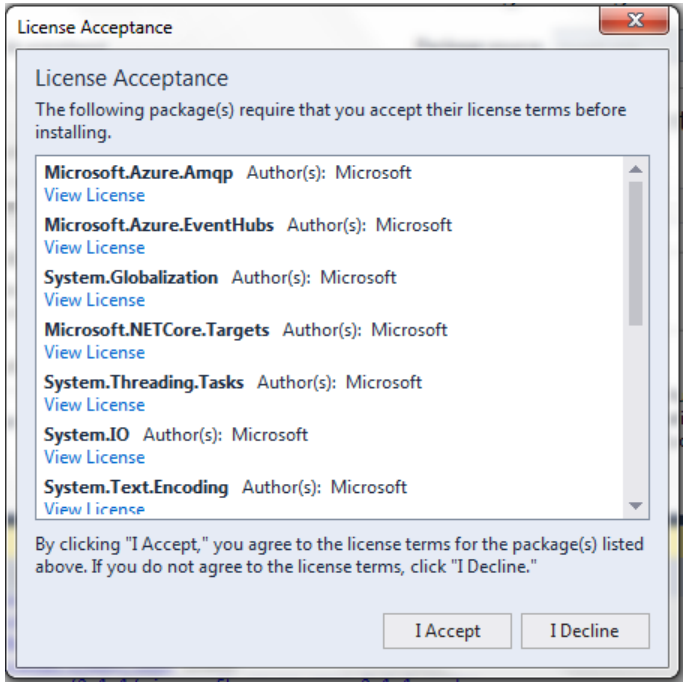
- Right Click on your new project and select ‘Manage NuGet Packages’
- Click Browse and search for Microsoft.Azure.EventHubs and then select to install it





Add the Event Hubs NuGet package

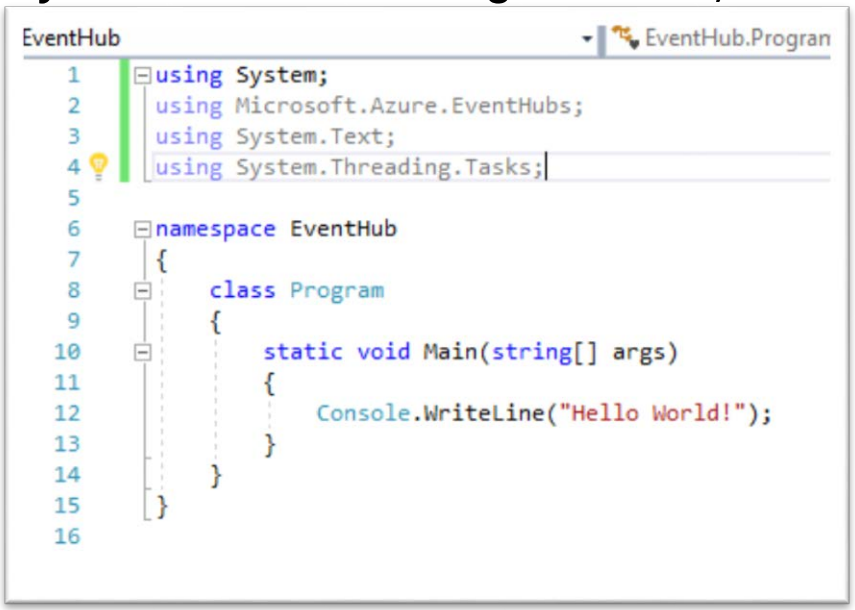
- You may be asked to preview the steps – select OK and then select I Accept for the installation of the License Acceptance



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

Example with 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";
```

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 scena
        // 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 SendMessagesToEventHub

```
// 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.");
}
```

Sending Messages to your Event Hub

Program.cs

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/andreaseh";

// 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

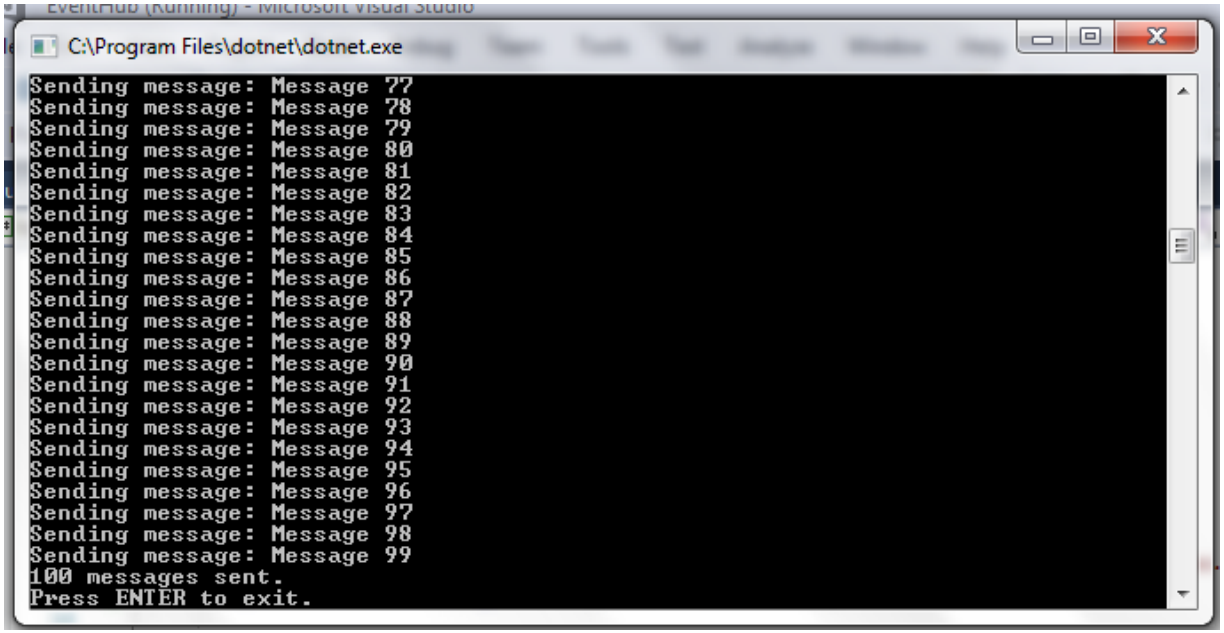
- Add the following code to the main of your Program Class

```
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!");
        }
    }
}
```

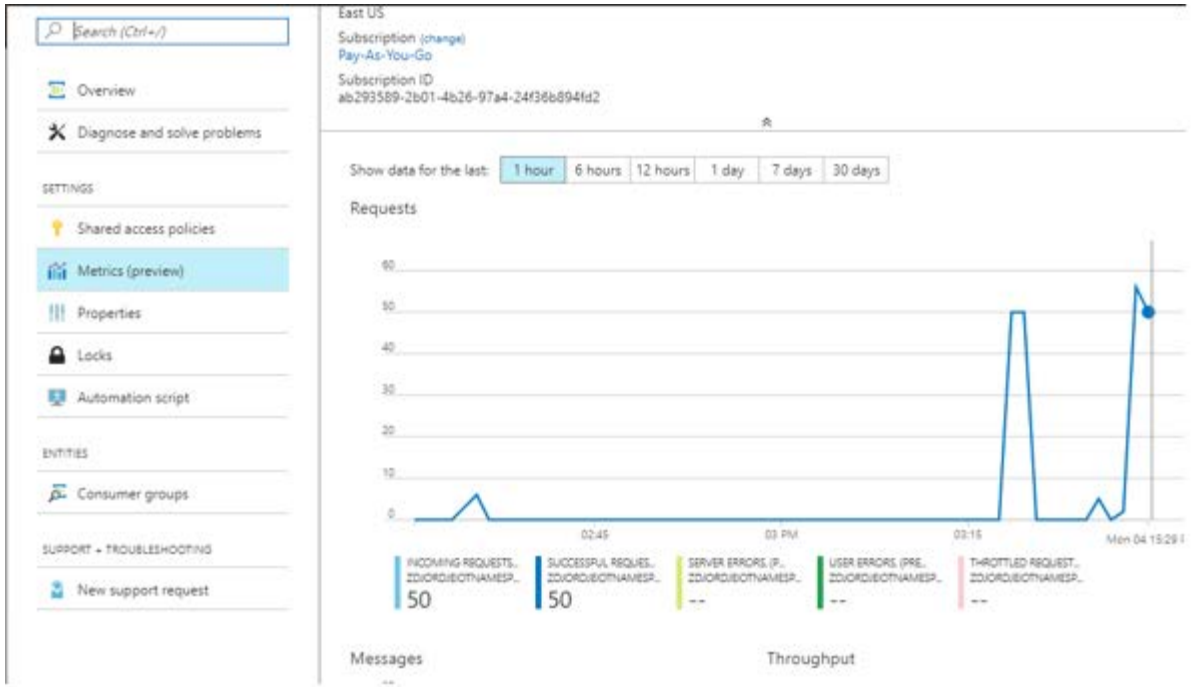
Run your program!

- 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.



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

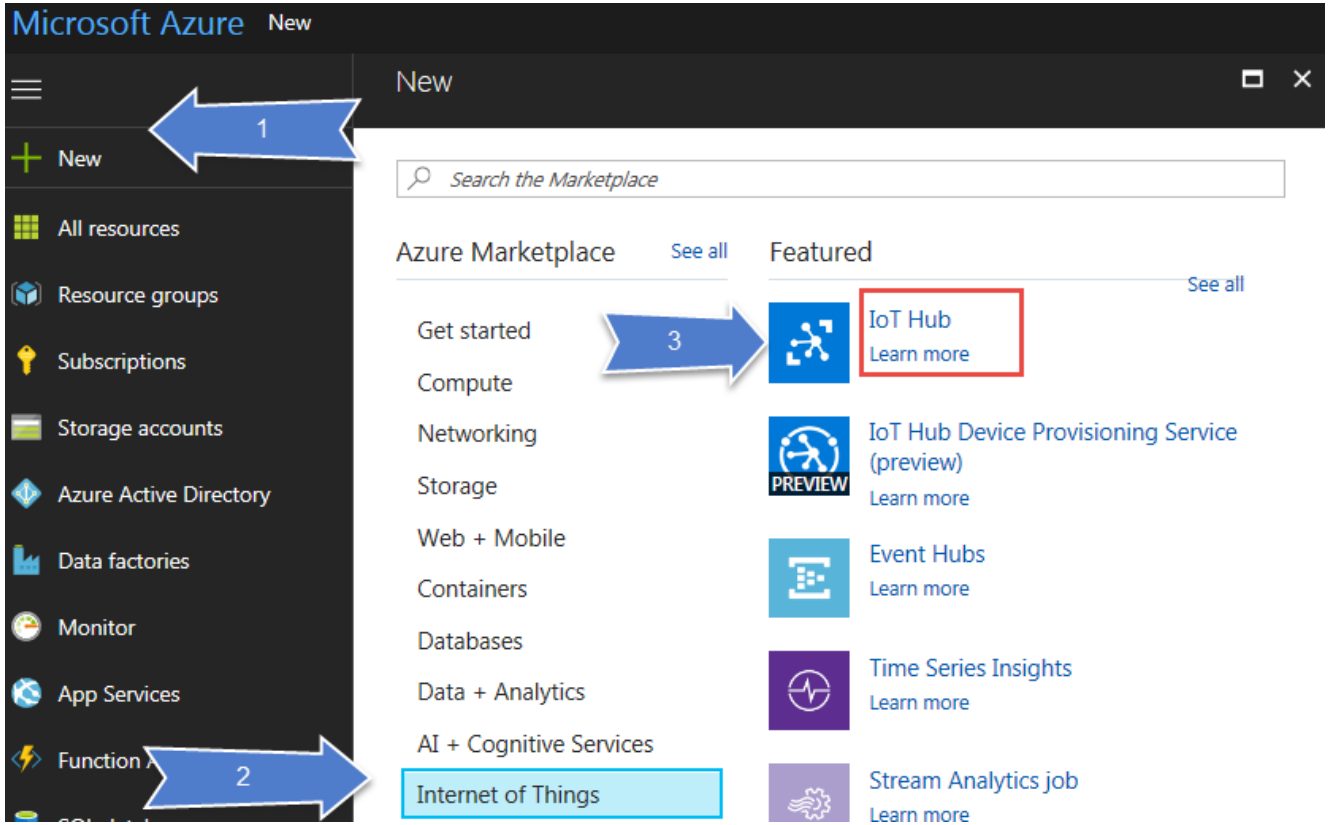
IoT Hubs SDKs

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

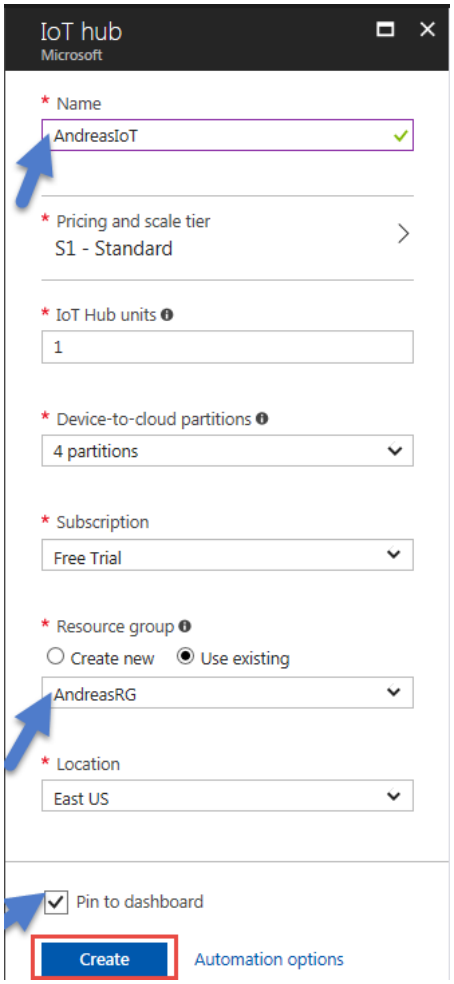
Creating an IoT Hub

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



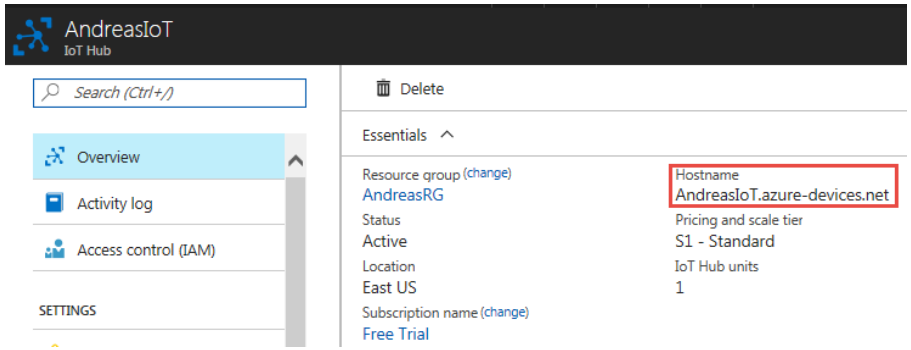
Creating an IoT Hub

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

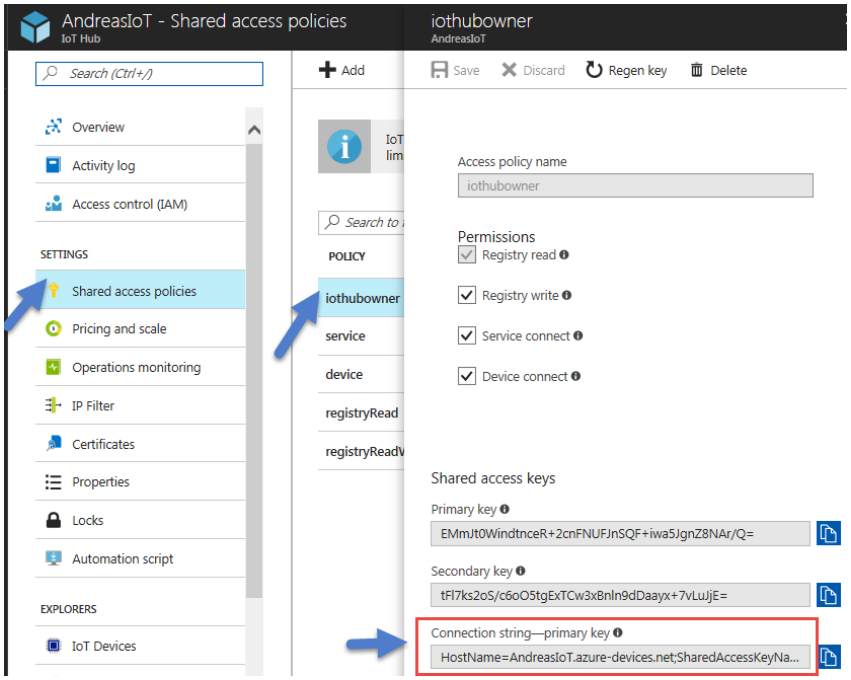


Creating an IoT Hub

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

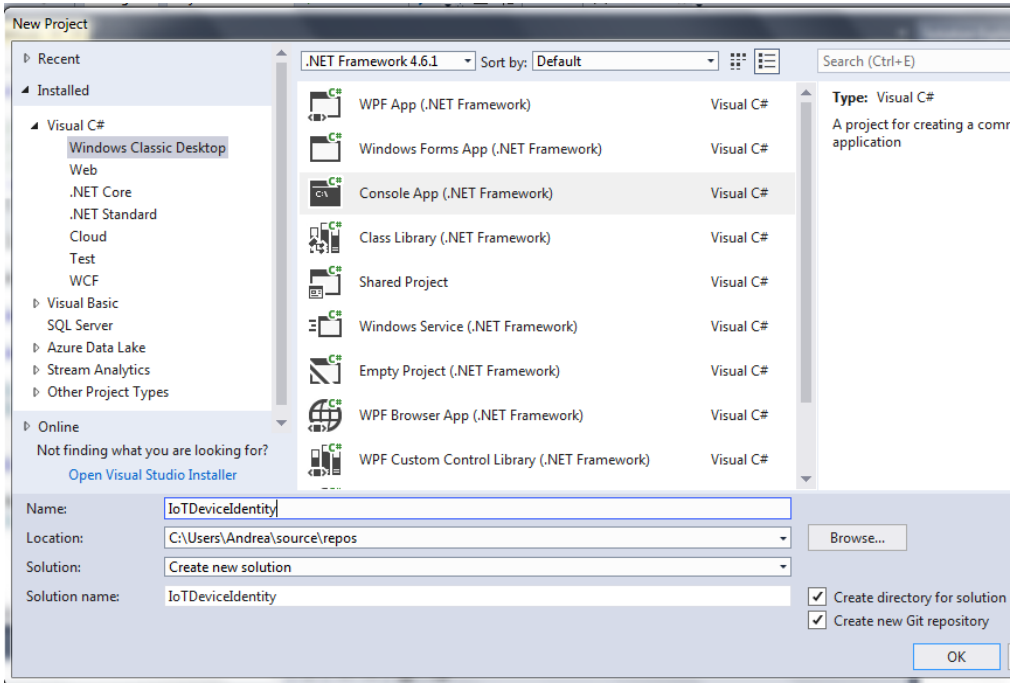


- Under your Shared access policies, select the iothubowner and copy your connection string, again storing it on a notepad



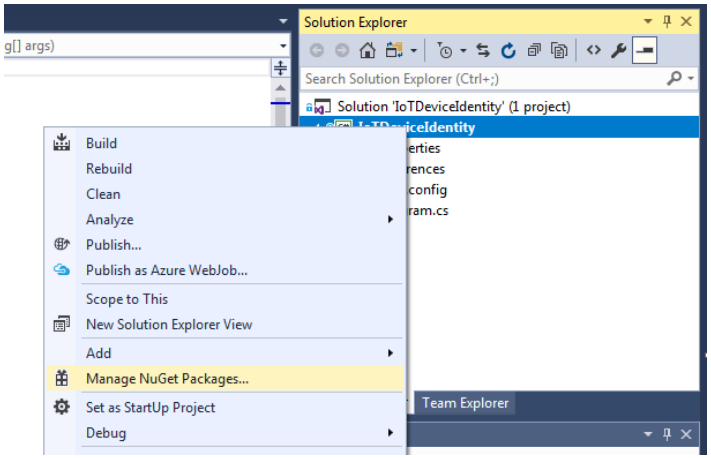
Creating an Identity

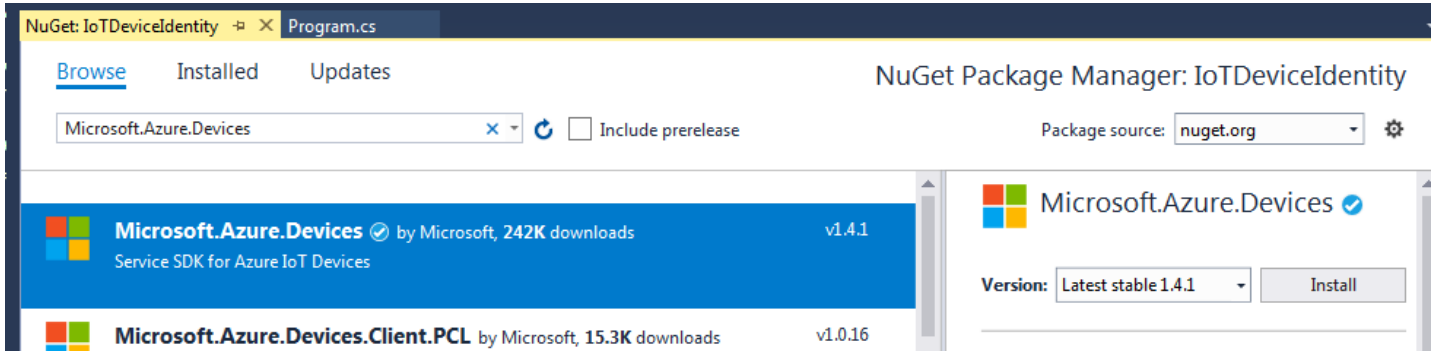
- In VS, Select File -> New -> Project
- Select the Windows Desktop section then Console App
- Add a Name and a Solution Name
- Note: You will work out of this solution for the entire IoT messaging



Creating an Identity

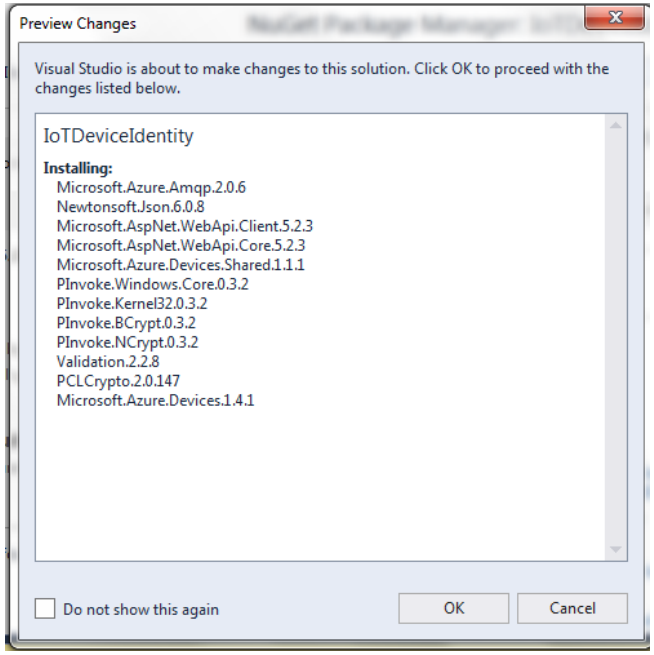
- Right click on your project and select Manage NuGet Packages
- Select Browse and search for **Microsoft.Azure.Devices** and select to Install it

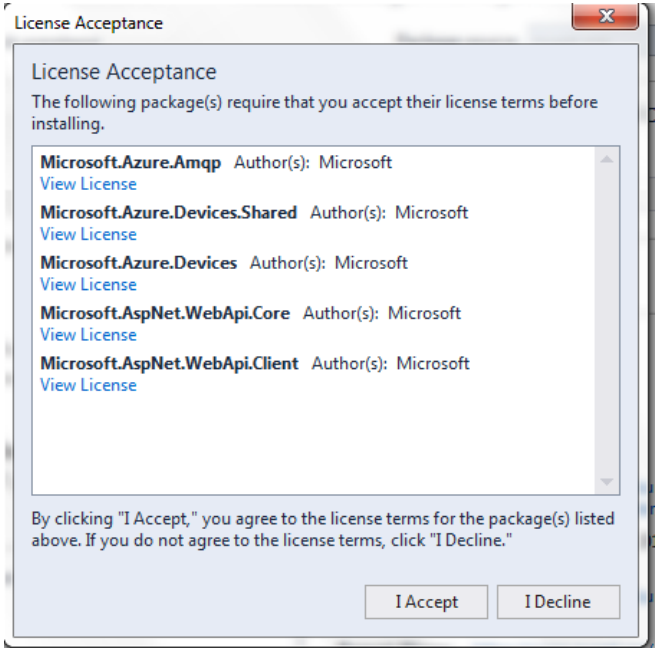




Creating an Identity

- 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;
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
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;SharedAcces
```

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

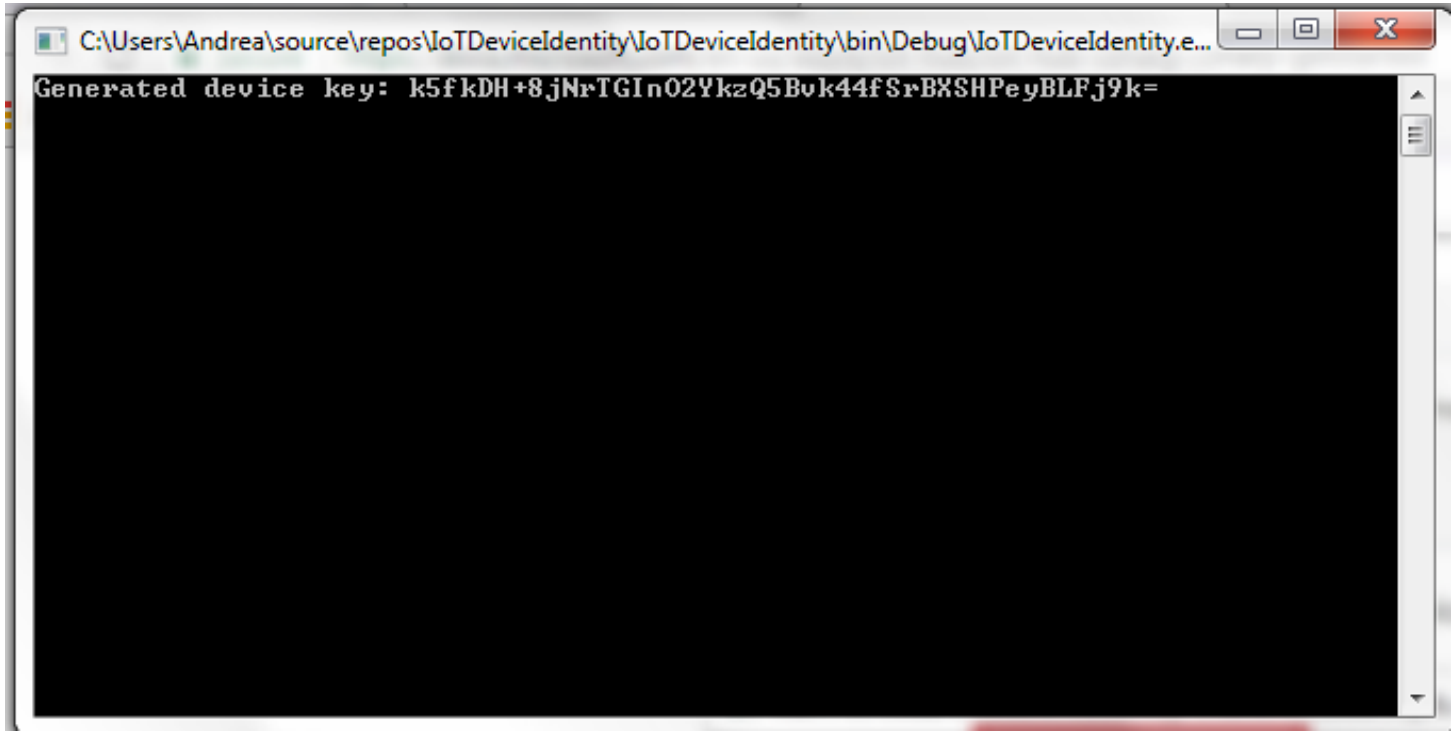
- Add the following in the main method

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

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

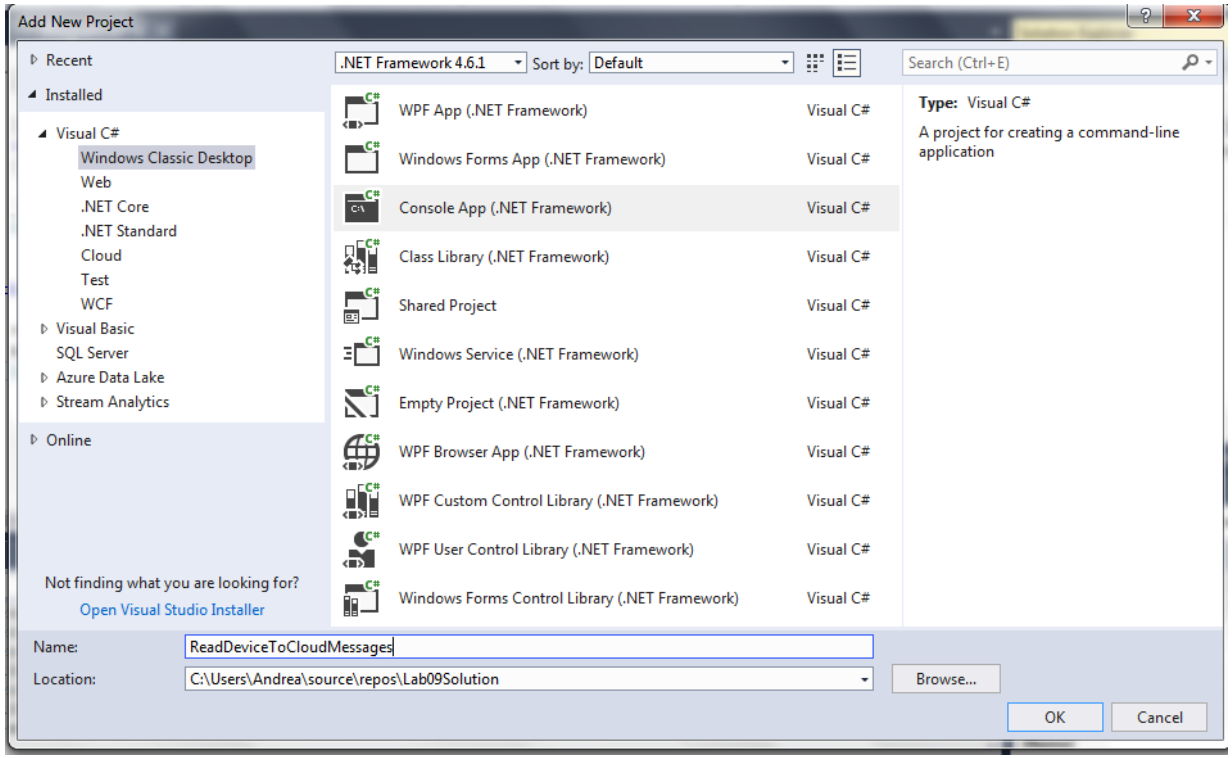
Run the application!

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



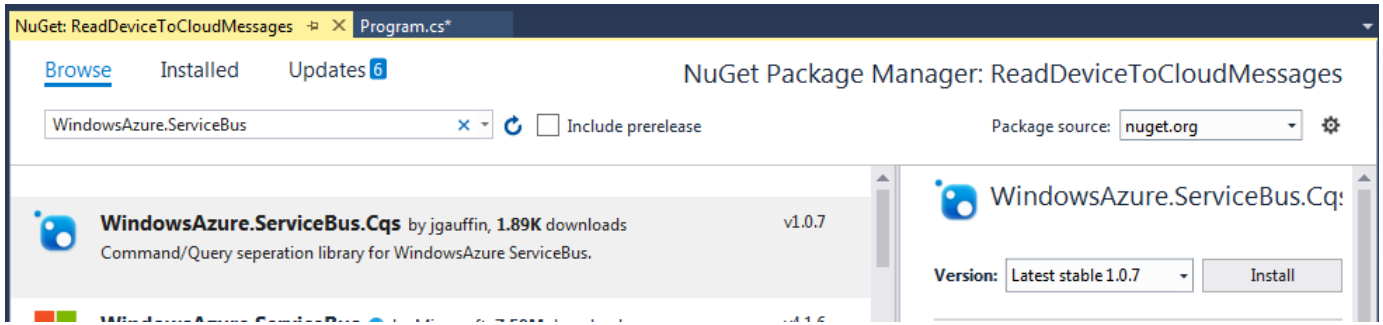
Working with device-to-cloud messages

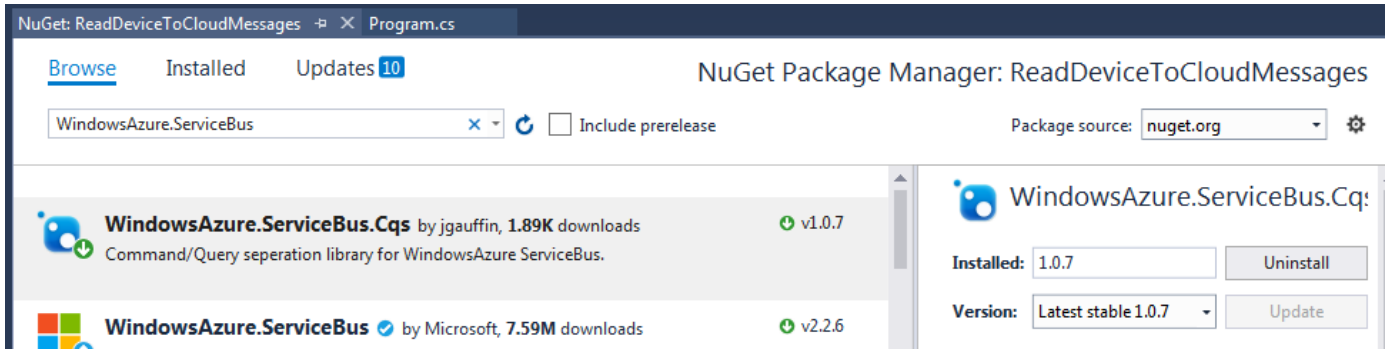
- In VS, select File -> Add -> New Project
- Select Windows Classic Desktop -> Console App
- Add a name and select OK



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

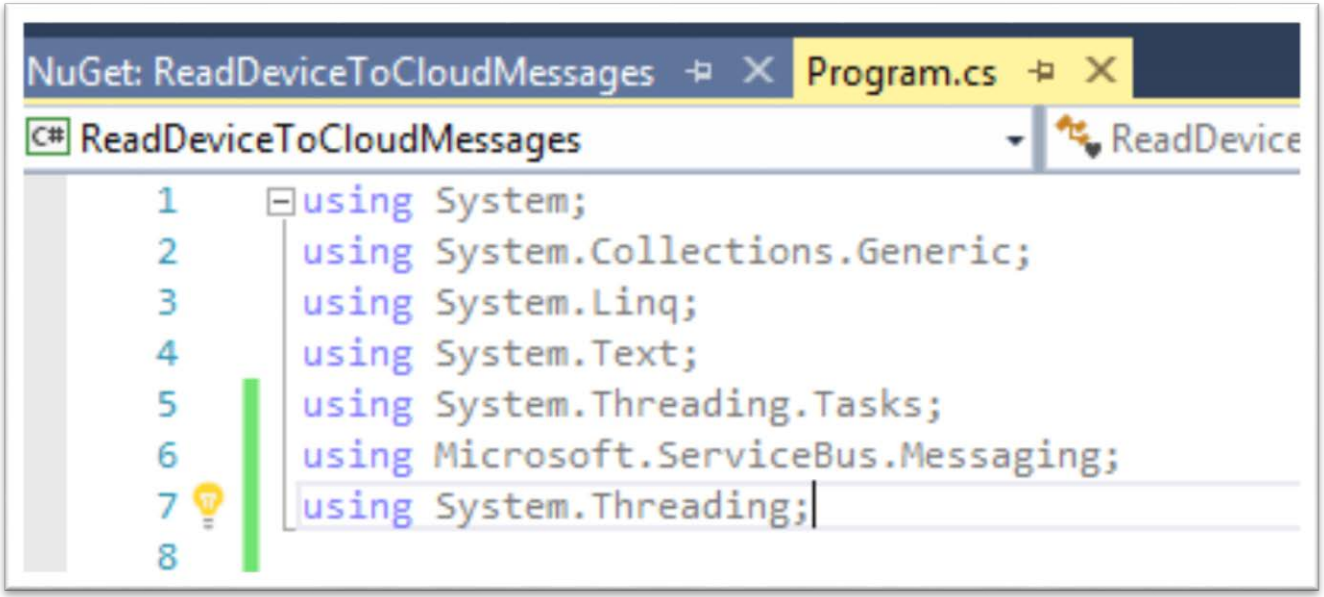




Working with device-to-cloud messages

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

```
using Microsoft.ServiceBus.Messaging;
using System.Threading;
```



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:

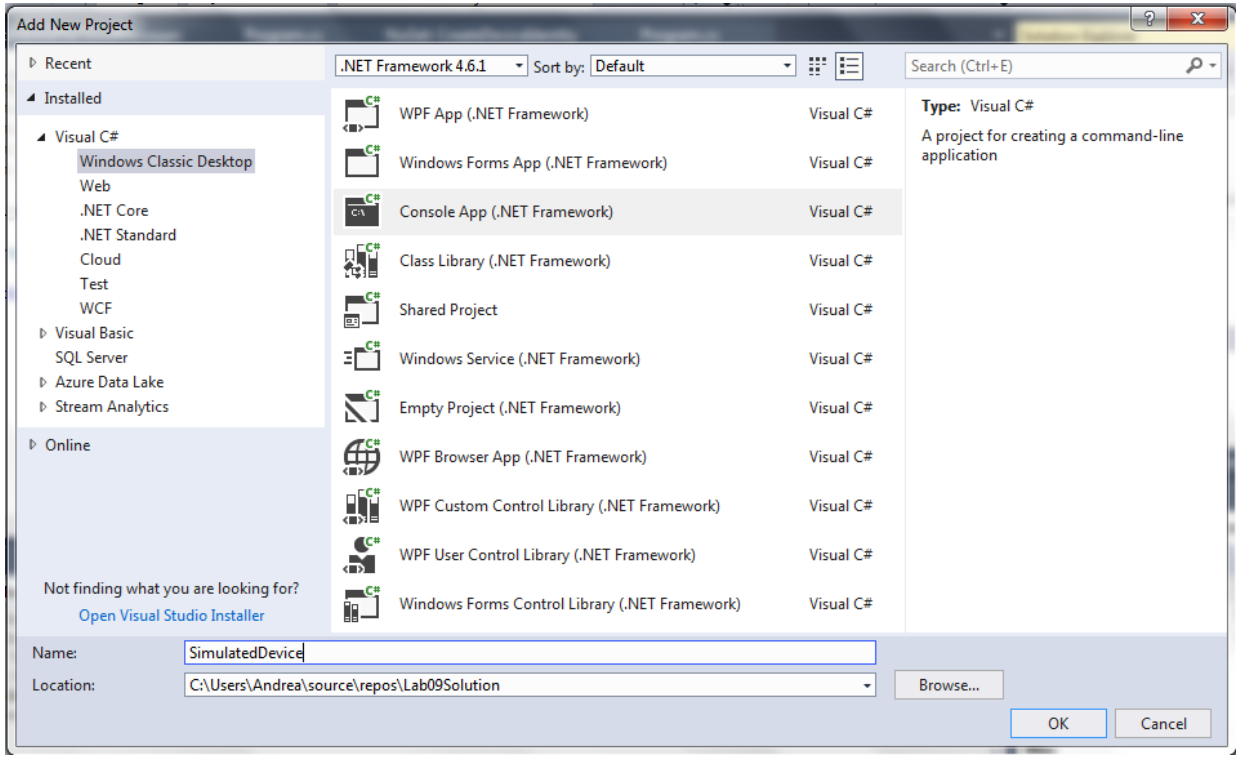
```
Console.WriteLine("Receive messages. Ctrl-C to exit.\n");
eventHubClient =
EventHubClient.CreateFromConnectionString(connectionString,
iotHubD2cEndpoint);
var d2cPartitions =
eventHubClient.GetRuntimeInformation().PartitionIds;
CancellationTokenSource cts = new CancellationTokenSource();
System.Console.CancelKeyPress += (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());
}
```

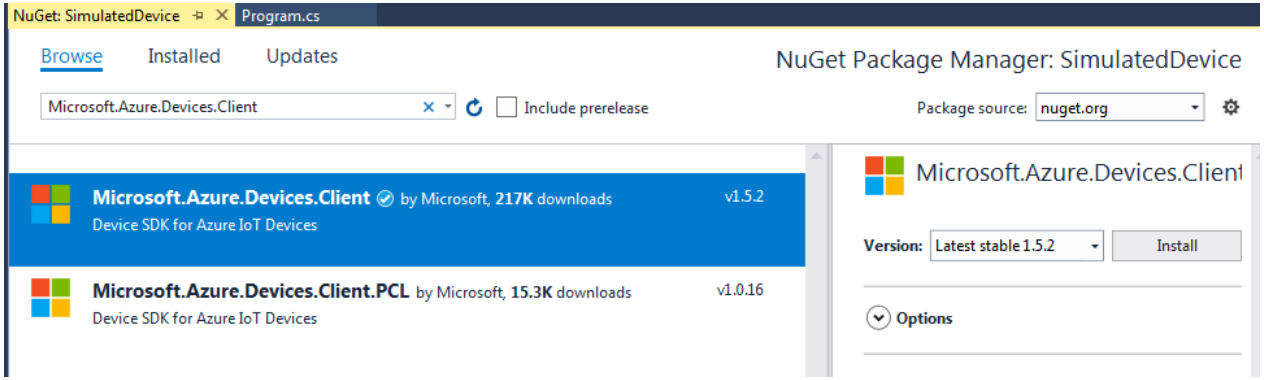
Create a device app

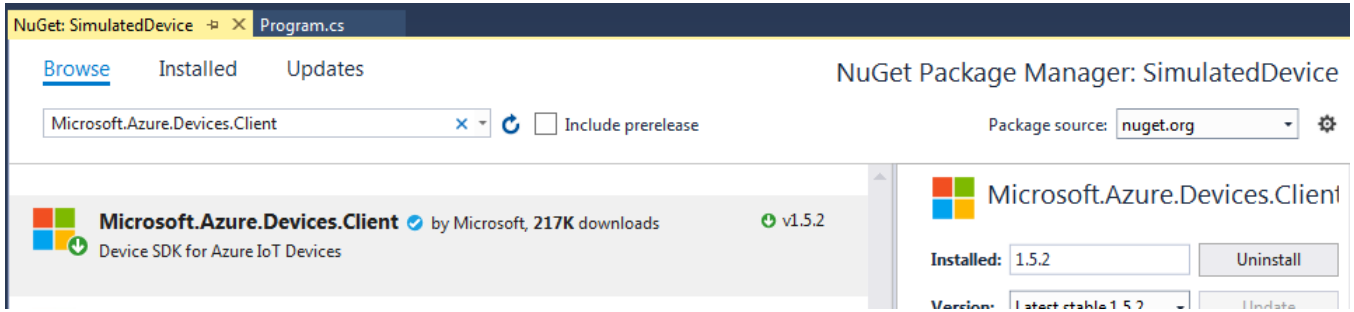
- In VS, select File -> Add -> New Project
- Select Windows Classic Desktop -> Console App
- Add a name and select OK



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

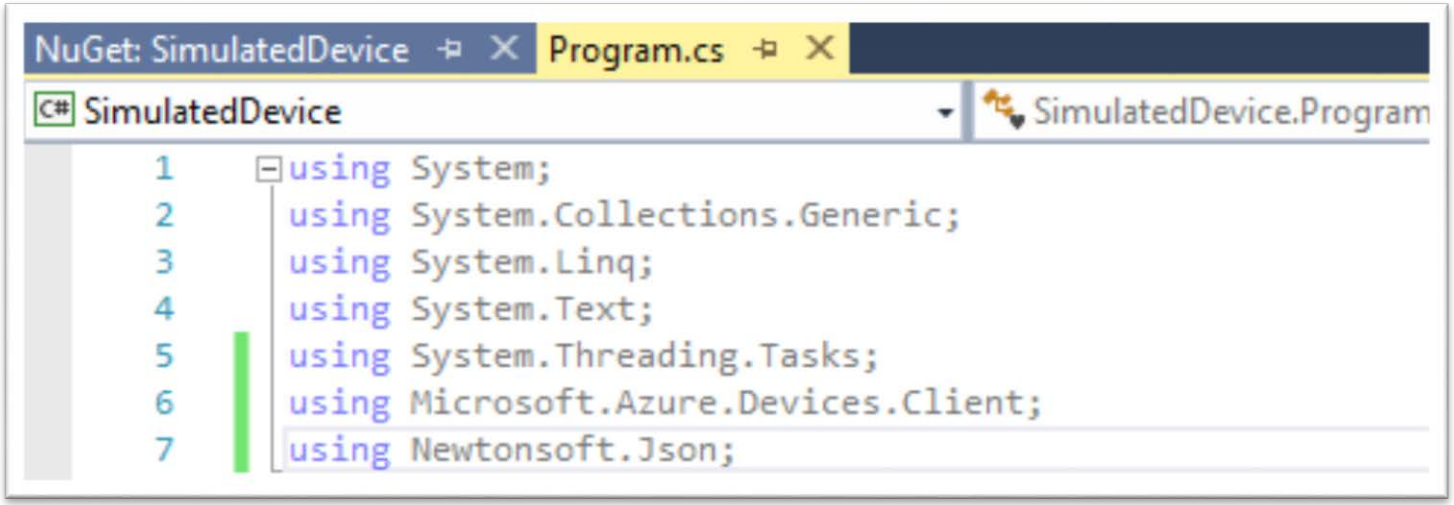




Create a device app

- 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+8jNrTGIn02YkzQ5Bvk44fSrBXSHPeyBLFj9k=";

    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

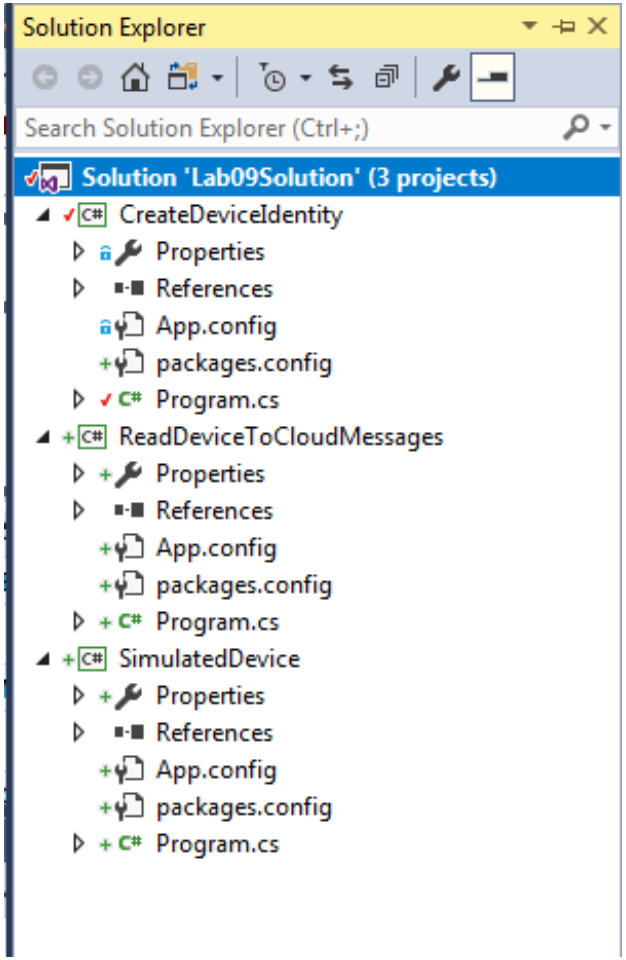
- Add the following to your Main method

```
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();
}
```

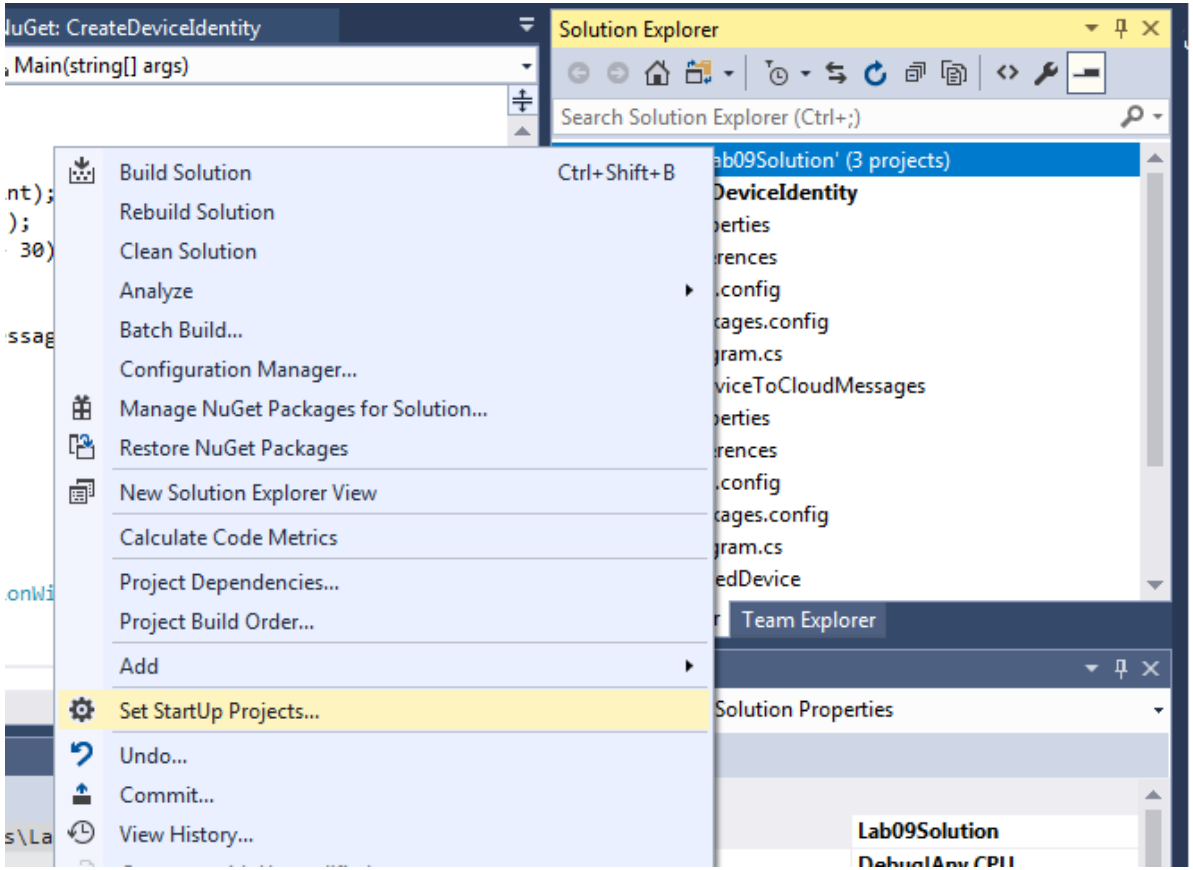
Your Solution

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



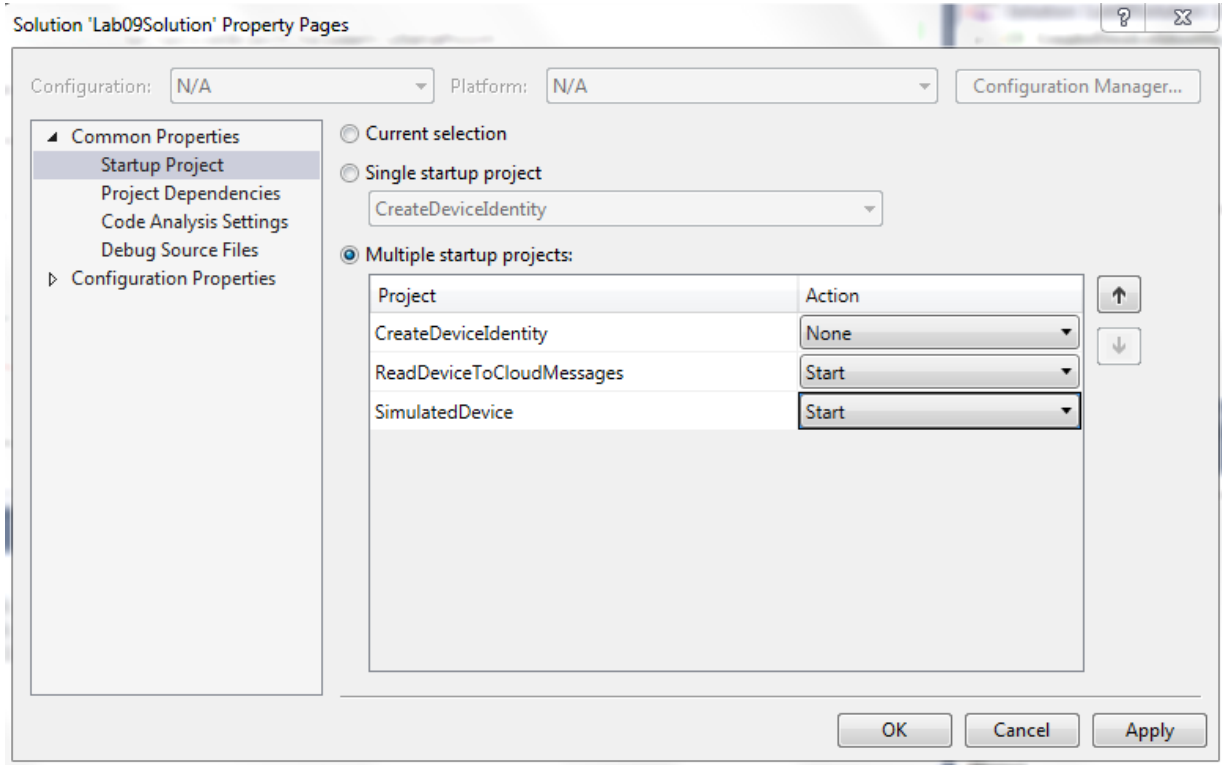
Running your app

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



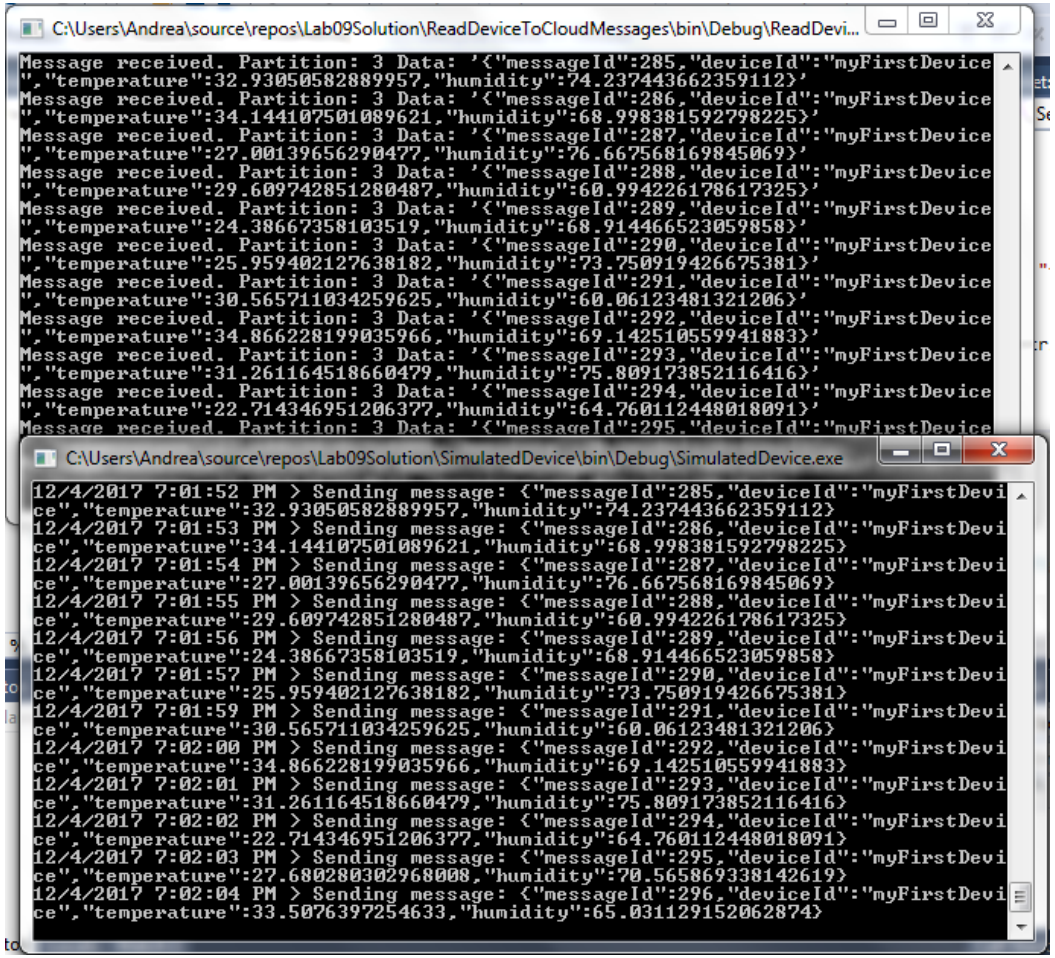
Running your app

- 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



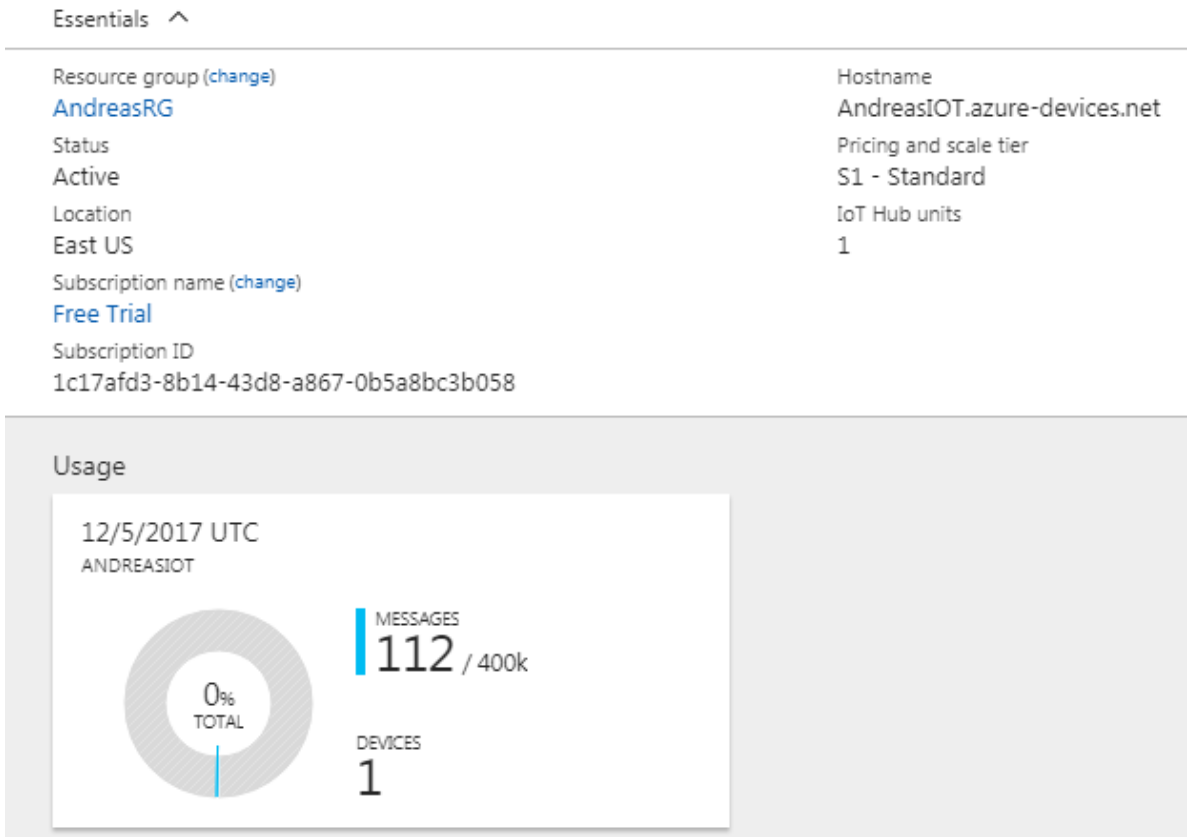
Running your app

- 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



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