Data Analysis using Hadoop: Module 6, Lesson 5  
Getting Started with Event Hubs Hands-On Lab

## Overview

In this lab, you will create an Event Hubs in Azure, and send message to Event Hubs using sender program written by C language.

## Objectives

In this hands-on lab you will learn how to:

* How to provision the Event Hub in Azure portal
* How to send messages to Event Hubs

## Prerequisites

The following are required to complete this hands-on lab:

* A Microsoft Azure subscription
* Microsoft Visual Studio Community Edition for Windows
* You should have completed [Module 6 Lessons 1-5](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module6/Lessons) and [Lesson 4 Lab](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module6/Labs)

Note: The Azure portal is continually improved and changed. The steps in this exercise reflect the user interface of the Microsoft Azure portal at the time of writing, but may not match the latest design of portal.

## Exercises

This hands-on lab includes the following exercises:

* Exercise 1: Create an Event Hub
* Exercise 2: Send message to Event Hubs
* Exercise 3: Create a Storage Account
* Exercise 4: Receive messages with EventProcessorHost
* Exercise 5: Run the applications

## Introduction

## Event Hubs is a service that processes large amounts of event data (telemetry) from connected devices and applications. After you collect data into Event Hubs, you can store the data using a storage cluster or transform it using a real-time analytics provider. This large-scale event collection and processing capability is a key component of modern application architectures including the Internet of Things (IoT).

## This tutorial shows how to use the Azure classic portal to create an Event Hub. It also shows you how to collect messages into an Event Hub using a console application written in C#, and how to retrieve them in parallel using the C# Event Processor Host library.

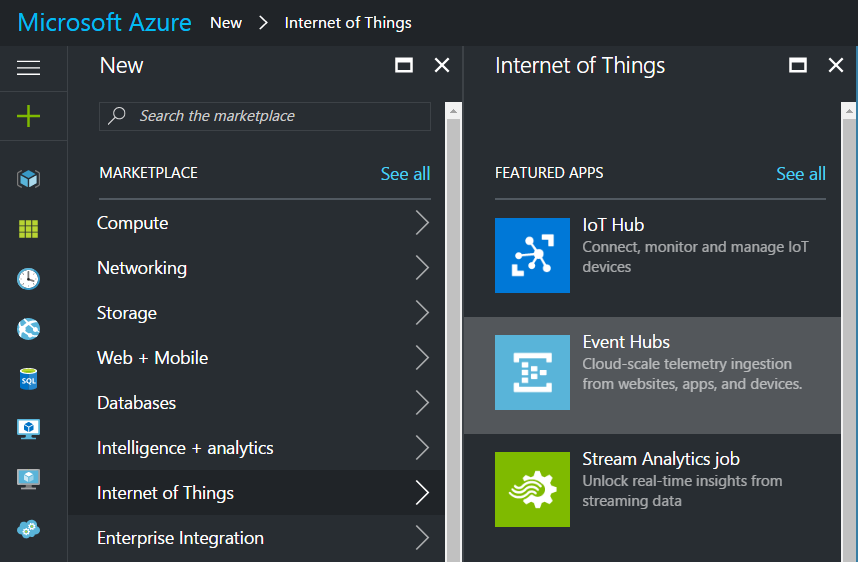
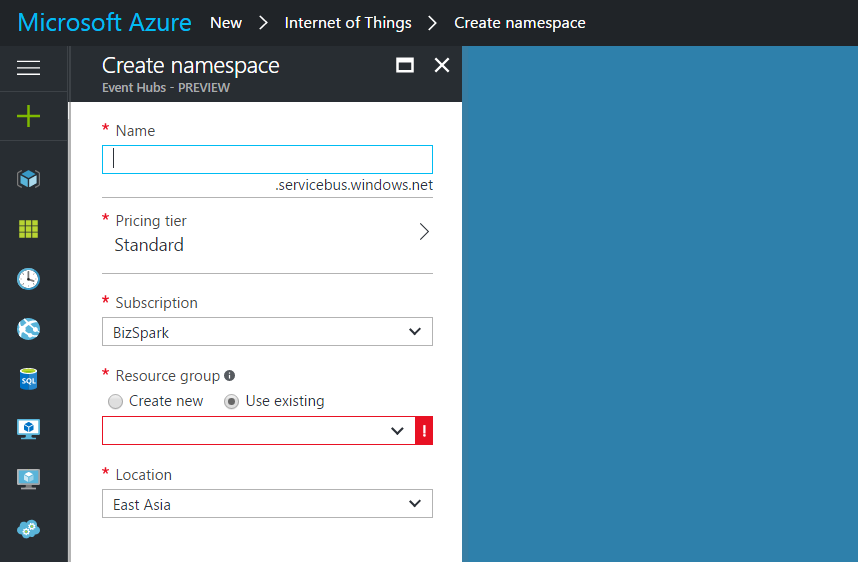
## Exercise 1: Create an Event Hub

The first task you have to perform is to creation of Event Hub.

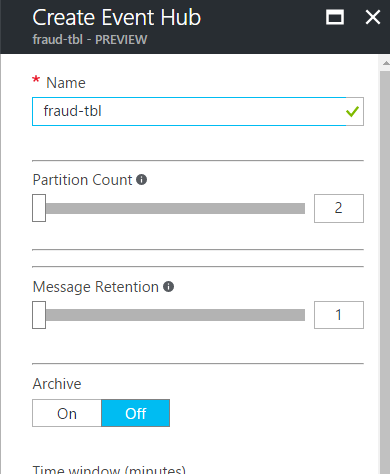
Students will generate events and push them to an Event Hub instance for real-time processing. Service Bus Event Hubs are the preferred method of event ingestion for Stream Analytics.

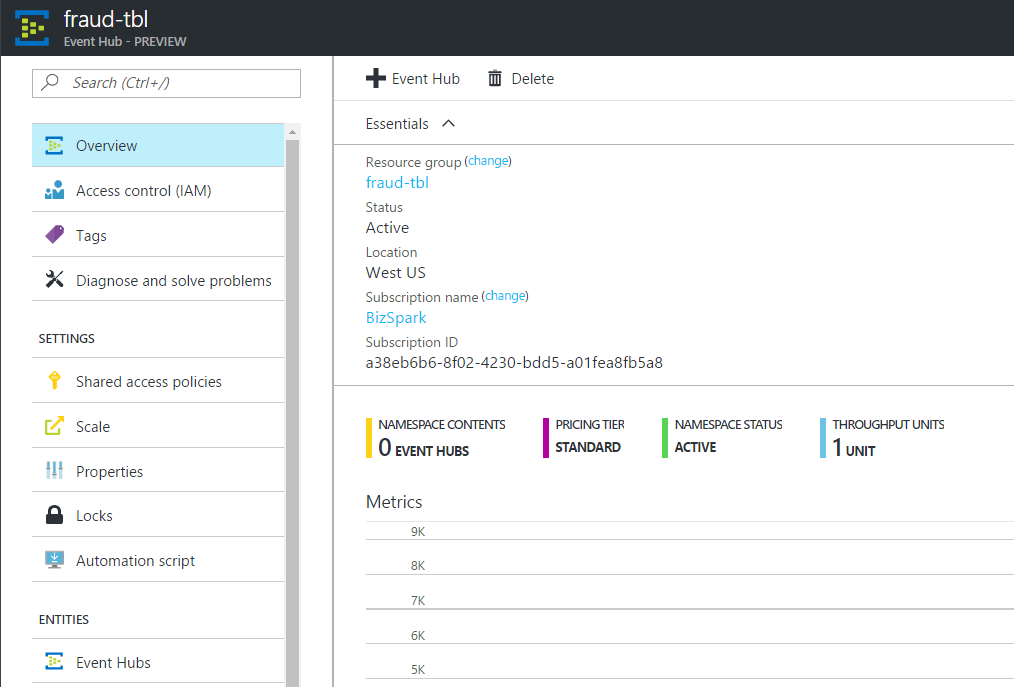
1. Create Event Hub Namespace

In the [Azure portal](https://portal.azure.com) click **New** > **Internet of Things** > **Event Hub** to create a new event hub namespace. Provide a friendly name for the namespace. Create or use an existing resource group. Resource groups allow you to group and manage resources together. Select an appropriate location (Region) to locate the Event Hub.



1. Create Event Hub

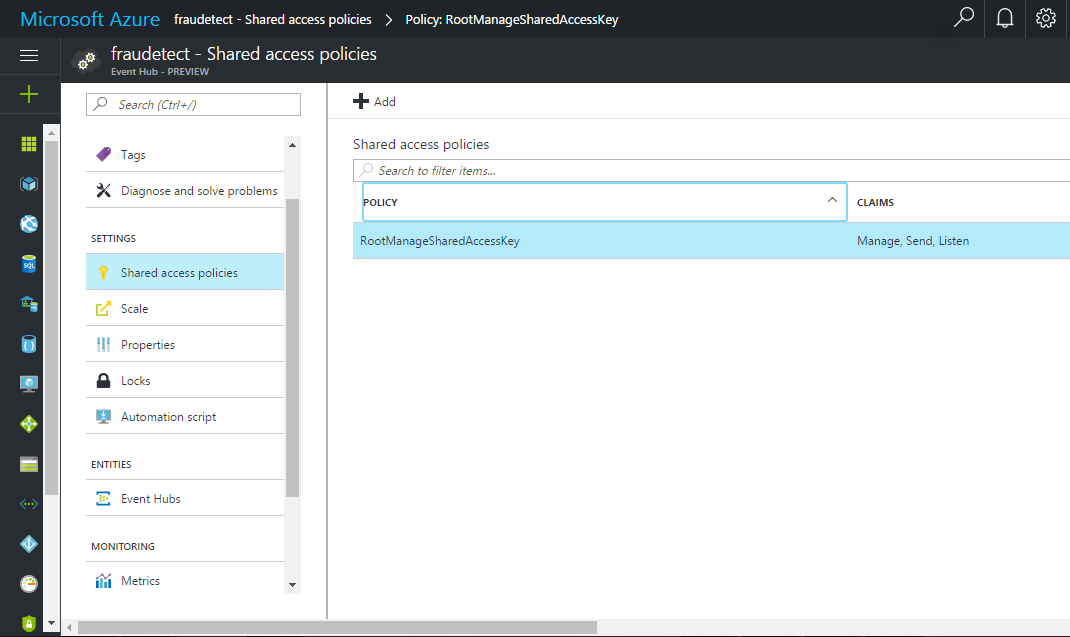
Click on **All resources** and select the newly created Event Hub from step 1.

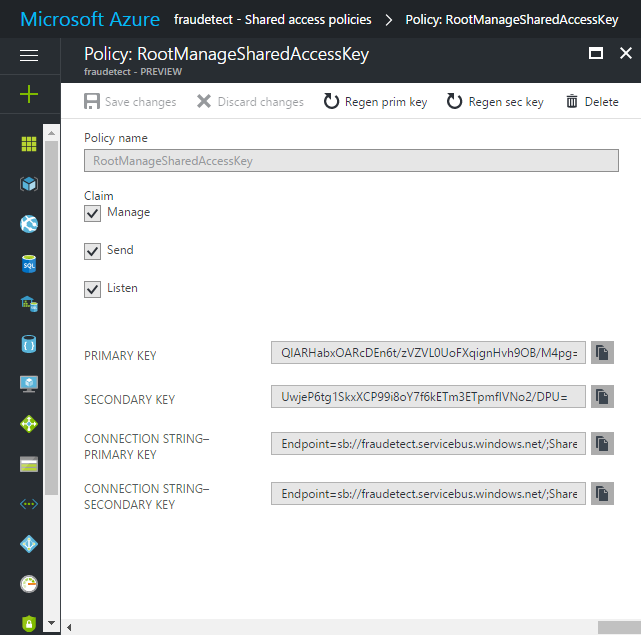


Click on the **+ Event Hub** and fill in a friendly name for the new Event Hub, the click on **Create** at the bottom of the page to complete the operation.

1. Copy **Connection Information**

Return to the Event Hub namespace and under settings click **Shared access policies**. Click on **RootManagerSharedAccessKey** and copy the Connection String – Primary Key.

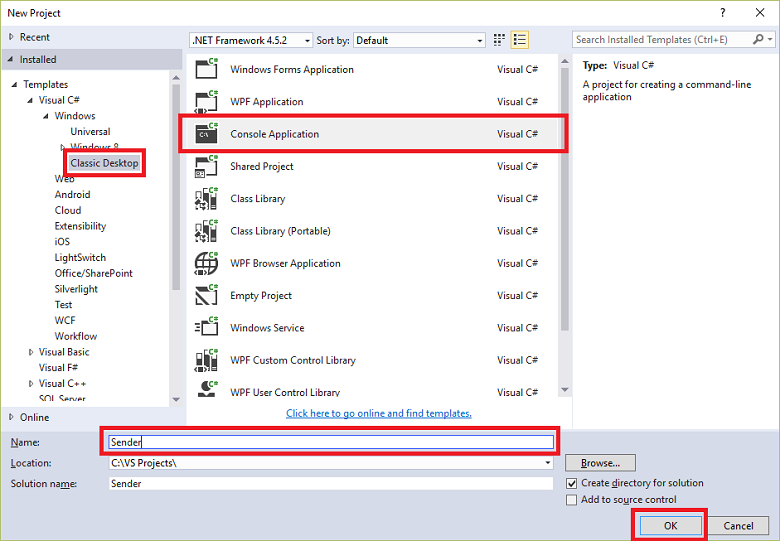




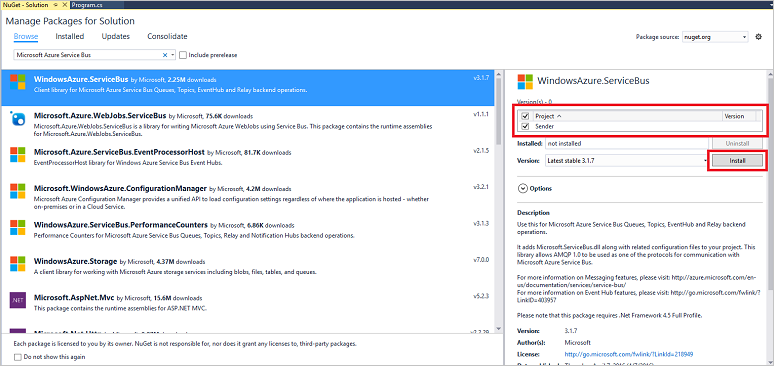
## Exercise 2: Send messages to Event Hubs.

In this section, you'll write a Windows console app that sends events to your Event Hub.

1. In Visual Studio, create a new Visual C# Desktop App project using the Console Application project template. Name the project Sender.



1. In Solution Explorer, right-click the solution, and then click Manage NuGet Packages for Solution.
2. Click the Browse tab, then search for "Microsoft.Azure.Service" Bus. Ensure that the project name (Sender) is specified in the Version(s) box. Click Install, and accept the terms of use.



This downloads, installs, and adds a reference to the Azure Service Bus library NuGet package.

Repeat step 3 for both "Microsoft.Azure.EventHubs" and "Microsoft.Azure.Ampq" packages.

1. Add the following using statement at the top of the Program.cs file:

using Microsoft.Azure.EventHubs;

1. Add the following fields to the Program class, substituting the placeholder values with the name of the Event Hub you created in the previous section, and the namespace-level connection string you saved previously.

private static EventHubClient eventHubClient;

private const string EHconnectionString = "{connection string}";

private const string EHEntityPath = "{Event Hub name}";

The Event Hub Name is TBLEventHub and the connection string is the RootManageSharedAccessKey connection string that students saved earlier.

1. Add the following MainAsync method to the Program class:

MainAsync Creates an EventHubsConnectionStringBuilder object from a 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.

private static async Task MainAsync(string[] args)

{

var connectionStringBuilder = new EventHubsConnectionStringBuilder(EhConnectionString)

{

EntityPath = EhEntityPath

};

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

await SendMessagesToEventHub(1000);

await eventHubClient.CloseAsync();

Console.WriteLine("Press any key to exit.");

Console.ReadLine();

}

1. Add the following method to the program class:

Create an Event Hub client and sends 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.");

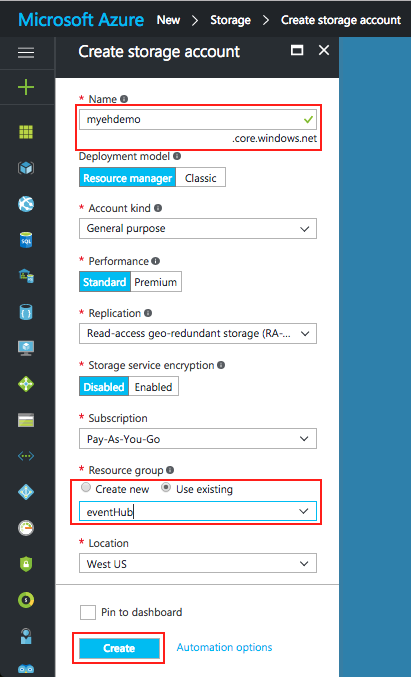
}

1. Finally, add the following line to the Main method:

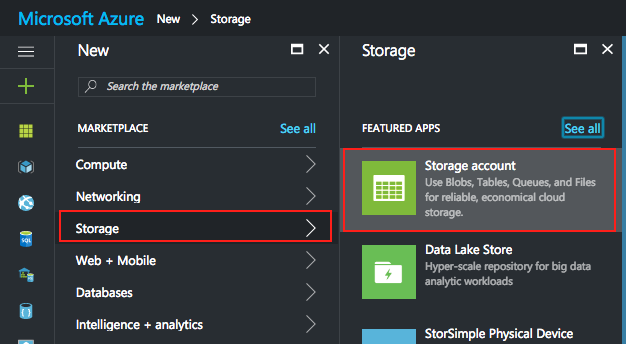
MainAsync(args).GetAwaiter().GetResult();

## Exercise 3: Create a Storage Account

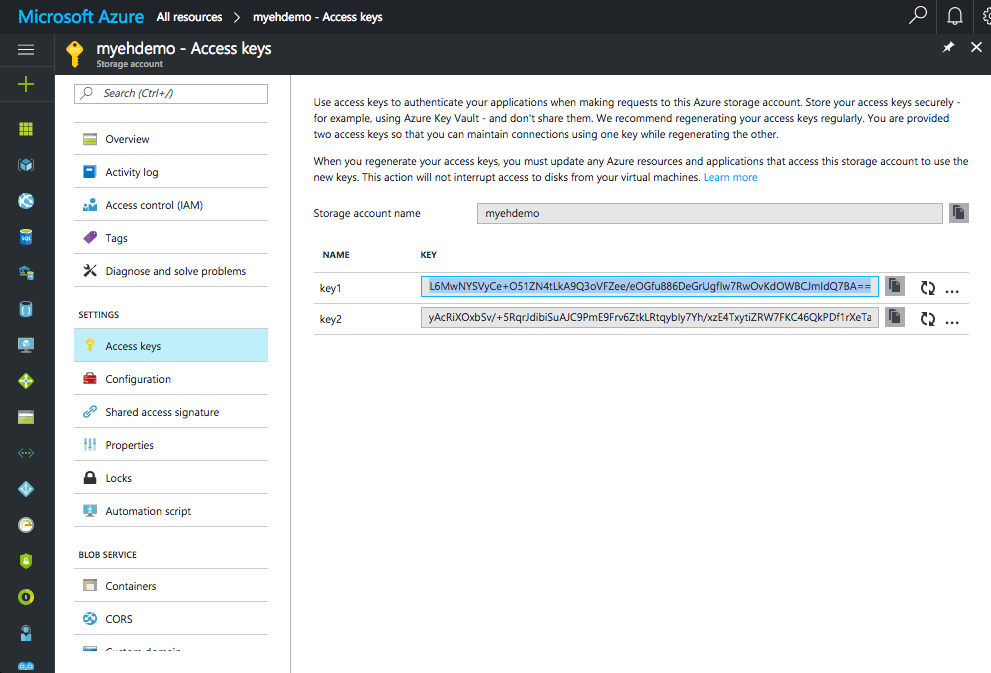
In order to receive events from the Event Hub, students need to a storage account.



1. Click NEW, and click “Storage”, then “Storage Account”, and then type a name for your storage account. Select your region and resource group, then click “Create”.



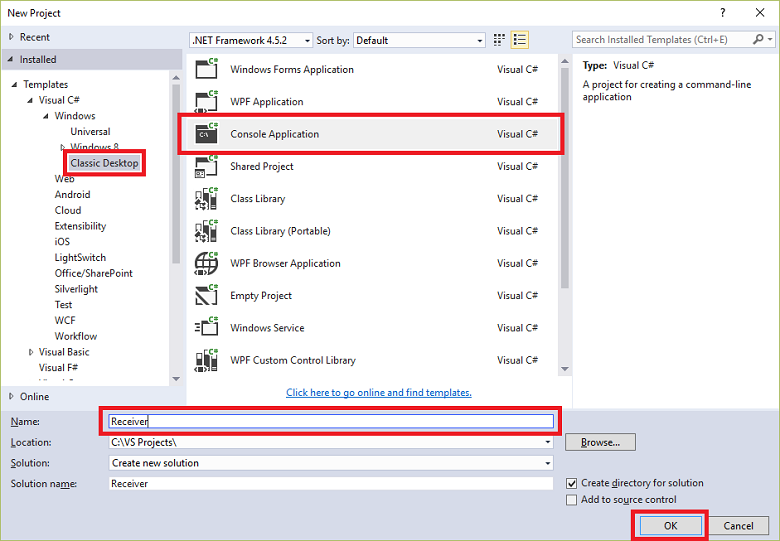
1. Click the newly created storage account, and then click “Access Keys”. Copy the primary access key to use later in this lab.



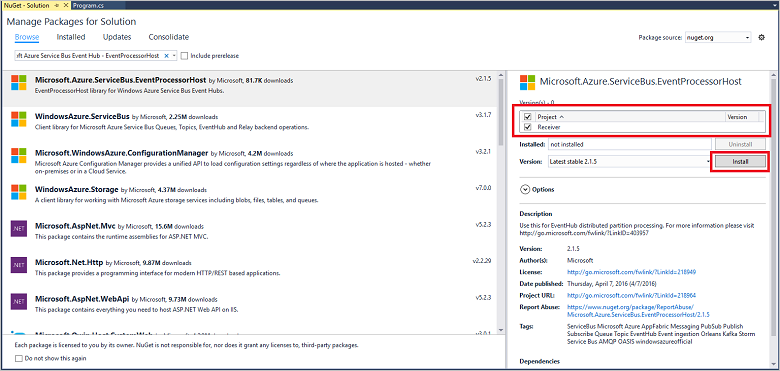
## Exercise 3: Receive Messages with EventProcessorHost

EventProcessorHost is a .NET class that simplifies receiving events from Event Hubs by managing persistent checkpoints and parallel receives from those Event Hubs. Using EventProcessorHost, you can split events across multiple receivers, even when hosted in different nodes. This example shows how to use EventProcessorHost for a single receiver.

1. In Visual Studio, create a new Visual C# Desktop App project using the Console Application project template. Name the project Receiver.



1. In Solution Explorer, right-click the solution, and then click Manage NuGet Packages for Solution.
2. Click the Browse tab, then search for "Microsoft.Azure.ServiceBus.EventProcessorHost. Ensure that the project name (Receiver) is specified in the Version(s) box. Click Install, and accept the terms of use.



This downloads, installs, and adds a reference to the Azure Service Bus Event Hub - EventProcessorHost NuGet package, with all its dependencies.

Repeat step 3 with the following packages:

"Microsoft.Data.Edm"

"Microsoft.Data.OData"

"Microsoft.Data.Services.Client"

"Microsoft.WindowsAzure.ConfigurationManager"

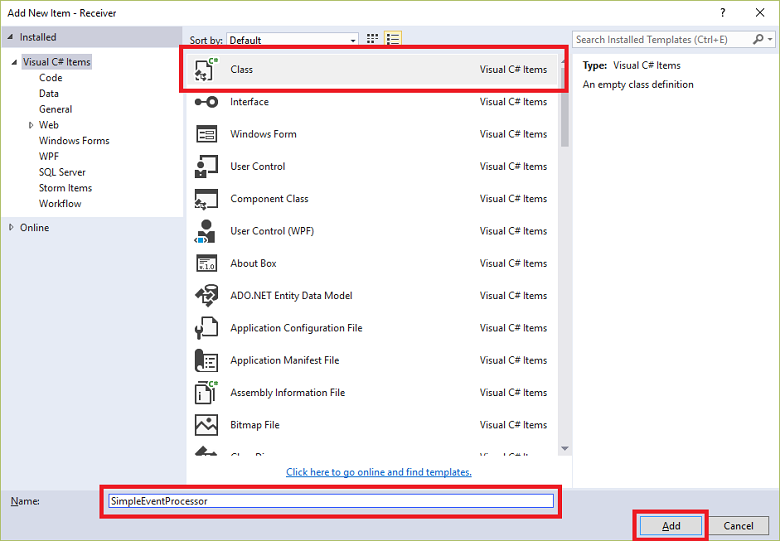
"Newtonsoft.Json"

"System.Spatial"

"WindowsAzure.ServiceBus"

"WindowsAzure.Storage"

1. Right-click the Receiver project, click Add, and then click Class. Name the new class SimpleEventProcessor, and then click Add to create the class.



1. Add the following statements at the top of the SimpleEventProcessor.cs file:

using Microsoft.ServiceBus.Messaging;

using System.Diagnostics;

1. Substitute the following code for the body of the class:

class SimpleEventProcessor : IEventProcessor

{

Stopwatch checkpointStopWatch;

async Task IEventProcessor.CloseAsync(PartitionContext context, CloseReason reason)

{

Console.WriteLine("Processor Shutting Down. Partition '{0}', Reason: '{1}'.", context.Lease.PartitionId, reason);

if (reason == CloseReason.Shutdown)

{

await context.CheckpointAsync();

}

}

Task IEventProcessor.OpenAsync(PartitionContext context)

{

Console.WriteLine("SimpleEventProcessor initialized. Partition: '{0}', Offset: '{1}'", context.Lease.PartitionId, context.Lease.Offset);

this.checkpointStopWatch = new Stopwatch();

this.checkpointStopWatch.Start();

return Task.FromResult<object>(null);

}

async Task IEventProcessor.ProcessEventsAsync(PartitionContext context, IEnumerable<EventData> messages)

{

foreach (EventData eventData in messages)

{

string data = Encoding.UTF8.GetString(eventData.GetBytes());

Console.WriteLine(string.Format("Message received. Partition: '{0}', Data: '{1}'",

context.Lease.PartitionId, data));

}

//Call checkpoint every 5 minutes, so that worker can resume processing from 5 minutes back if it restarts.

if (this.checkpointStopWatch.Elapsed > TimeSpan.FromMinutes(5))

{

await context.CheckpointAsync();

this.checkpointStopWatch.Restart();

}

}

}

This class will be called by the EventProcessorHost to process events received from the Event Hub. Note that the SimpleEventProcessor class uses a stopwatch to periodically call the checkpoint method on the EventProcessorHost context. This ensures that, if the receiver is restarted, it will lose no more than five minutes of processing work.

1. In the Program class, add the following using statement at the top of the file:

using Microsoft.ServiceBus.Messaging;

Then, replace the Main method in the Program class with the following code, substituting the Event Hub name and the namespace-level connection string that you saved previously, and the storage account and key that you copied in the previous sections.

static void Main(string[] args)

{

string eventHubConnectionString = "{Event Hub connection string}";

string eventHubName = "{Event Hub name}";

string storageAccountName = "{storage account name}";

string storageAccountKey = "{storage account key}";

string storageConnectionString = string.Format("DefaultEndpointsProtocol=https;AccountName={0};AccountKey={1}", storageAccountName, storageAccountKey);

string eventProcessorHostName = Guid.NewGuid().ToString();

EventProcessorHost eventProcessorHost = new EventProcessorHost(eventProcessorHostName, eventHubName, EventHubConsumerGroup.DefaultGroupName, eventHubConnectionString, storageConnectionString);

Console.WriteLine("Registering EventProcessor...");

var options = new EventProcessorOptions();

options.ExceptionReceived += (sender, e) => { Console.WriteLine(e.Exception); };

eventProcessorHost.RegisterEventProcessorAsync<SimpleEventProcessor>(options).Wait();

Console.WriteLine("Receiving. Press enter key to stop worker.");

Console.ReadLine();

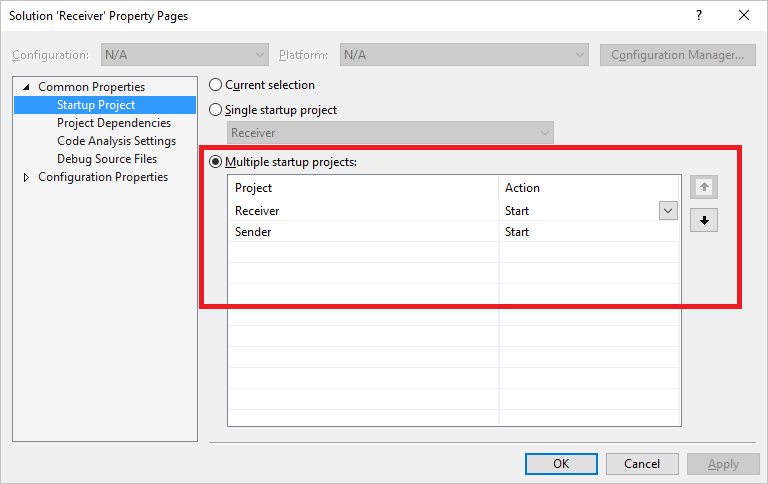
eventProcessorHost.UnregisterEventProcessorAsync().Wait();

}

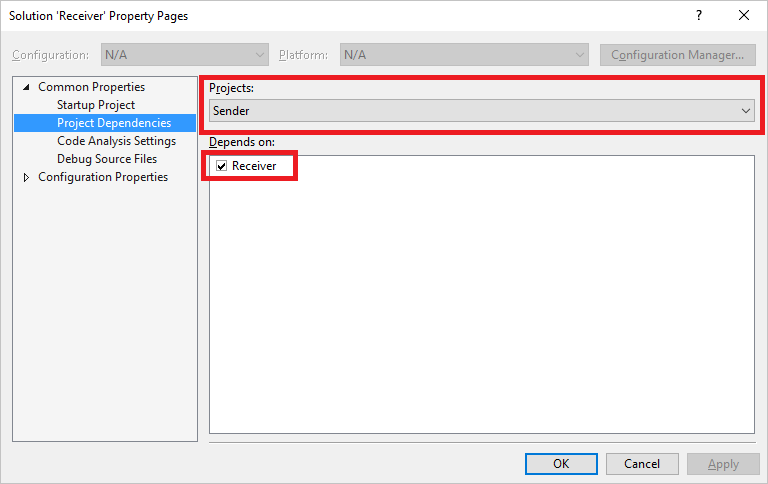
## Exercise 4: Run the applications

Now, run the applications:

1. From within Visual Studio, open the Receiver project you created earlier.
2. Right-click the Receiver solution, then click Add, and then click Existing Project.
3. Locate the existing Sender.csproj file, then double-click it to add it to the solution.
4. Again, right-click the Receiver solution and then click Properties. The Receiver property page is displayed.
5. Click Startup Project, then click the Multiple startup projects button. Set the Action box for both the Receiver and Sender projects to Start.



1. Click Project Dependencies. In the Projects box, click Sender. In the Depends on box, make sure Receiver is checked.



1. Click OK to dismiss the Properties dialog.
2. Press F5 to run the Receiver project from within Visual Studio, then wait for it to start the receivers for all the partitions.
3. The Sender project will run automatically. Press Enter in the console window, and see the events appear in the receiver window.

Press Ctrl+C in the Sender window to end the Sender application, then press Enter in the Receiver window to shut down that application.

\*\*\*There may be an aggregateException thrown after running for 30s\*\*\*

In this case, the receiver window will not log reception of the messages, but you may view messages in the Azure Portal in your event hub dashboard.

## Summary

In this hands-on lab, you learned how to:

* How to provision the Event Hub in Azure portal.
* How to send messages to Event Hubs.
* How to provision a Storage Account in Azure portal.
* How to receive messages at the Event Hubs.