**Lab Manual- Azure Message Based Solutions**

**Prepared for**: TechPledge

**Date:** 18th Nov 2020

**Prepared by:** Aditi Srivastava

Document Name: Lab Manual

**Document Number** AZ-MSG-204

**Contributor:**

Table of Contents

[1 OBJECTIVE 3](#_Toc55405023)

[2 PRE-REQUISISTE 4](#_Toc55405024)

[3 Setup Up Service Bus queues 4](#_Toc55405025)

[3.1 **Create a Service Bus Namespace and Queue** 4](#_Toc55405026)

[3.2 **Create a queue in the Service Bus** 5](#_Toc55405027)

[4 Setup Up Service Bus queues 7](#_Toc55405028)

[4.2 **Create a Create a console application to send message** 7](#_Toc55405029)

[4.3 **Create Connmection Stric** 8](#_Toc55405030)

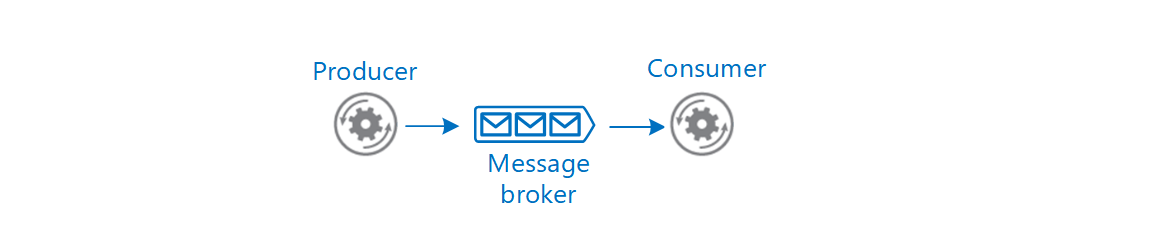
[5 Write code to send messages to the queue 9](#_Toc55405031)

[6 Receive messages from the queue 15](#_Toc55405032)

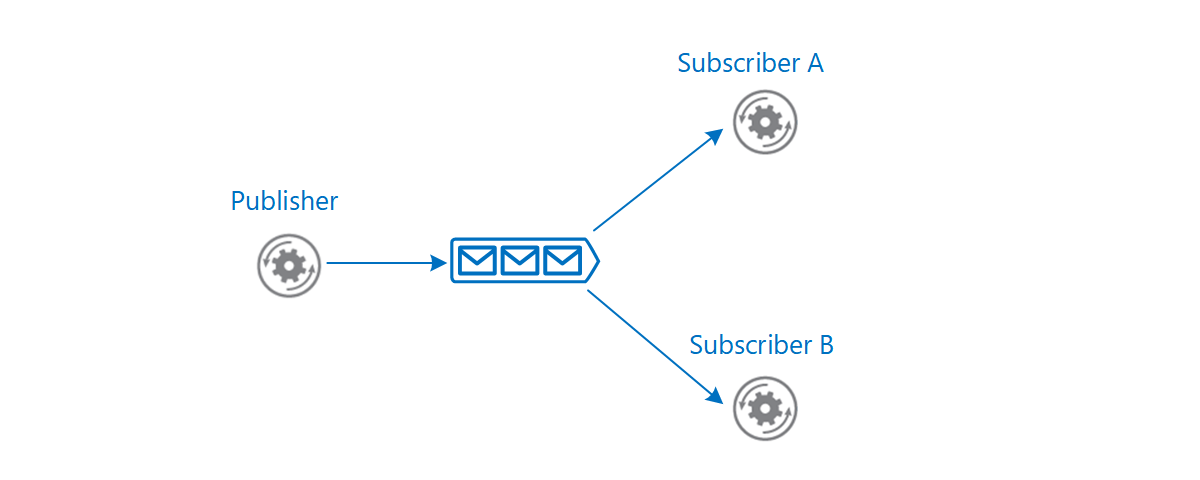
[6.1 Write code to receive messages from the queue 16](#_Toc55405033)

# OBJECTIVE

* This Lab describes the different types of messages and the entities that participate in a messaging infrastructure. Based on the requirements of each message type, the article recommends Azure messaging services. The options include [**Azure Service Bus**](https://docs.microsoft.com/en-us/azure/architecture/guide/technology-choices/messaging#azure-service-bus)**,**[**Event Grid**](https://docs.microsoft.com/en-us/azure/architecture/guide/technology-choices/messaging#azure-event-grid), and [**Event Hubs**](https://docs.microsoft.com/en-us/azure/architecture/guide/technology-choices/messaging#azure-event-hubs)**.**
* At an architectural level, a **message** is a datagram created by an entity (**producer),** to distribute information so that other entities (**consumers)** can be aware and act accordingly. The **producer** and the **consumer** can communicate directly or optionally through an intermediary entity (**message broker**)



* **Messages** can be classified into two main categories.
  + If the producer expects an action from the consumer, that message is a **command.**
  + If the message informs the consumer that an action has taken place, then the message is an **event.**
* A common pattern for implementing event messaging is the [**Publisher-Subscriber**](https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber) pattern.



What are Queues?

**A queue is simply a list of messages.** The messages flow from the bottom of the queue to the top of the queue in the order they were added to the queue. It is known by computer scientists as a **FIFO** data structure. FIFO stands for First In-First Out.

Azure Service Bus

[Azure Service Bus](https://docs.microsoft.com/en-us/azure/service-bus-messaging/) queues are well suited for transferring commands from producers to consumers

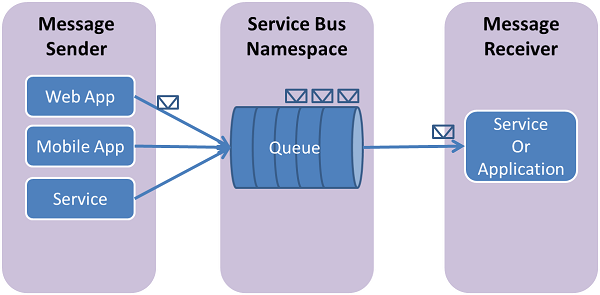
Dead-letter queue (DLQ)

A **Service Bus queue** has a default **subqueue**, called the [dead-letter queue (DLQ)](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dead-letter-queues) to hold **messages** that couldn't be delivered or processed. **Service Bus** or the **message processing logic** in the consumer can add **messages** to the DLQ. The DLQ keeps the **messages** until they are retrieved from the queue.

# PRE-REQUISISTE

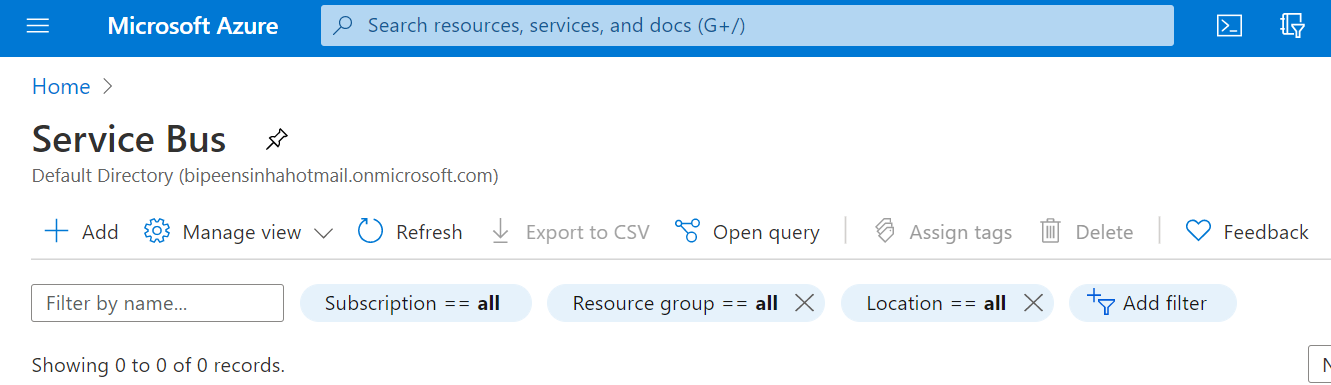
* Accounts in Azure
* A local Computer with 4 CPU, 16 GB RAM, 200 GB disk space

# Setup Up Service Bus queues

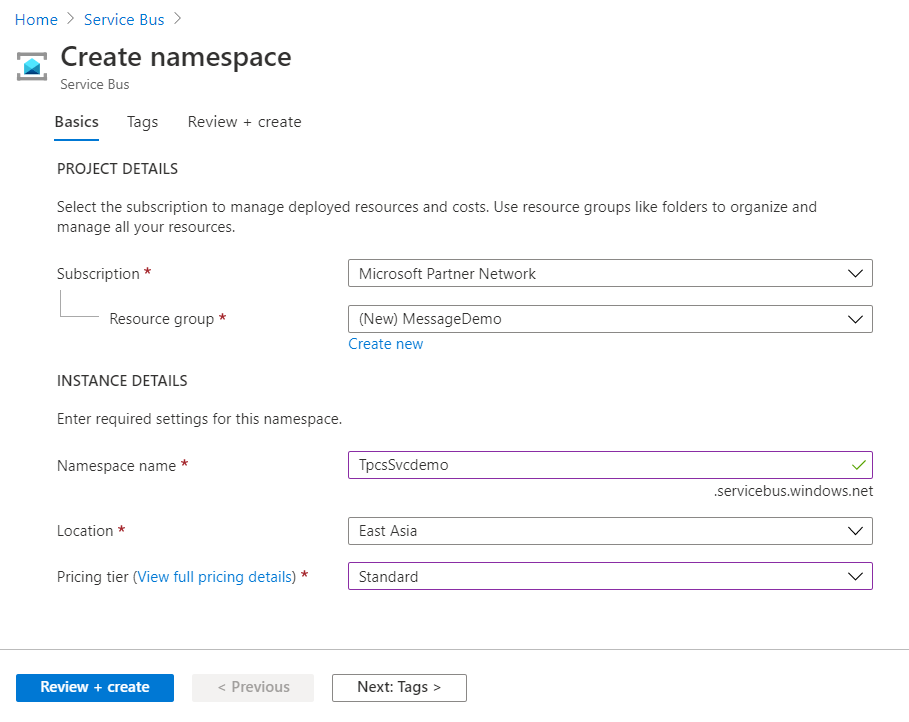


## **Create a Service Bus Namespace and Queue**

1. Sign in to the [Azure portal](https://portal.azure.com/)
2. In the left navigation pane of the portal, select  select **Service Bus**.



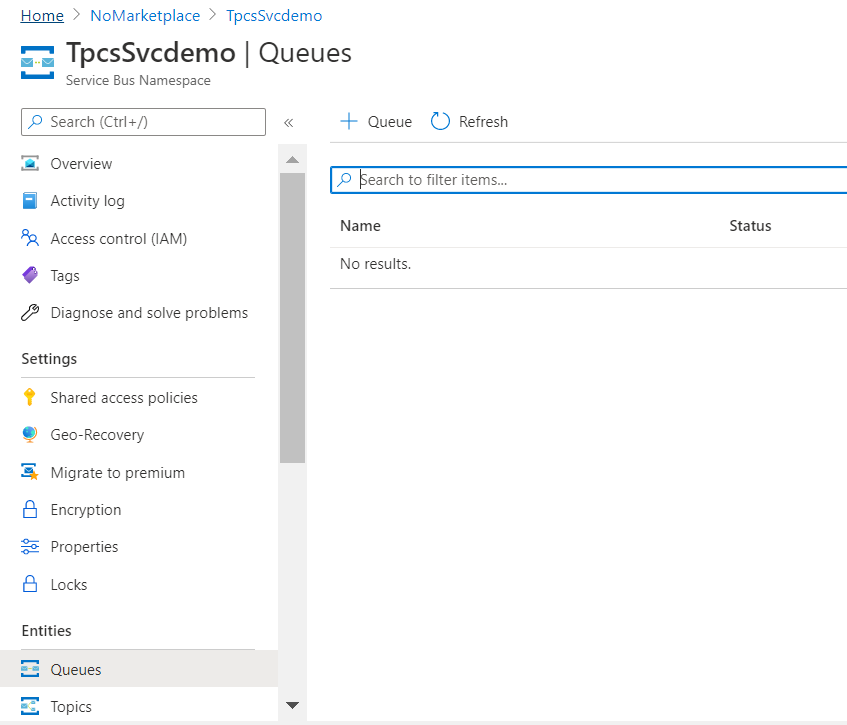
1. In the **Create namespace** dialog



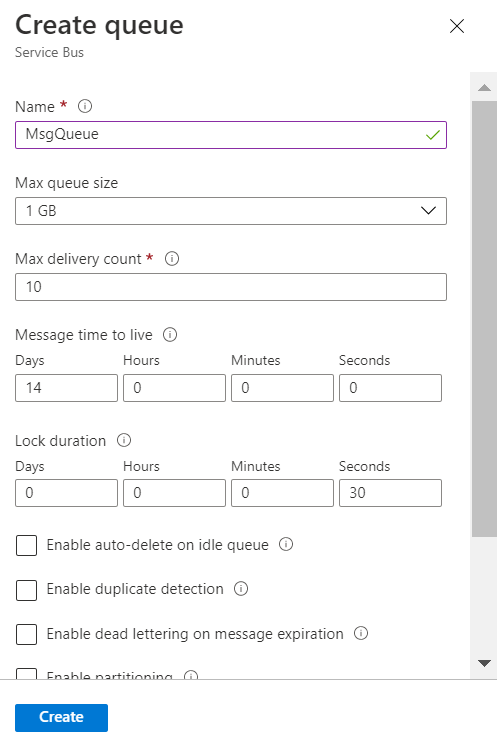
1. Verify the Service Bus get Created

## **Create a queue in the Service Bus**

1. On the **Service Bus Namespace** page, select **Queues** in the left navigational menu.
2. On the **Queues** page, select **+ Queue** on the toolbar.



1. Enter a **name** for the queue, and leave the other values with their defaults.
2. Now, select **Create**.



# Setup Up Service Bus queues

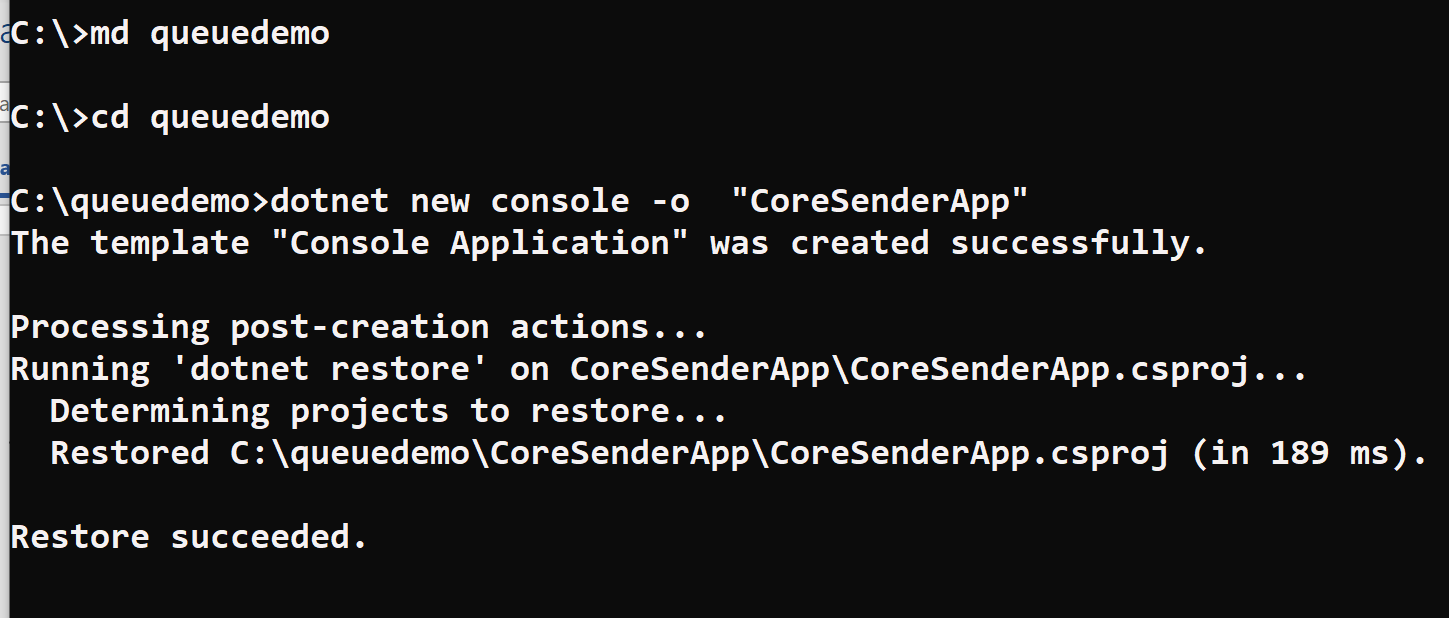
## **Create a Create a console application to send message**

1. Launch Visual Studio and create a **new Console App (.NET Core)**project for C#. This example names the app CoreSenderApp.

**md queuedemo**

**cd queuedemo**

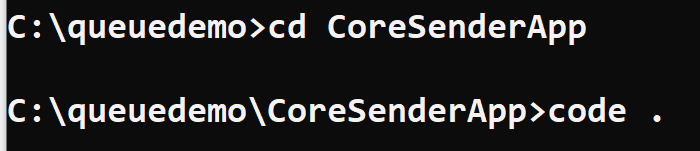
**dotnet new console -o “CoreSenderApp“**



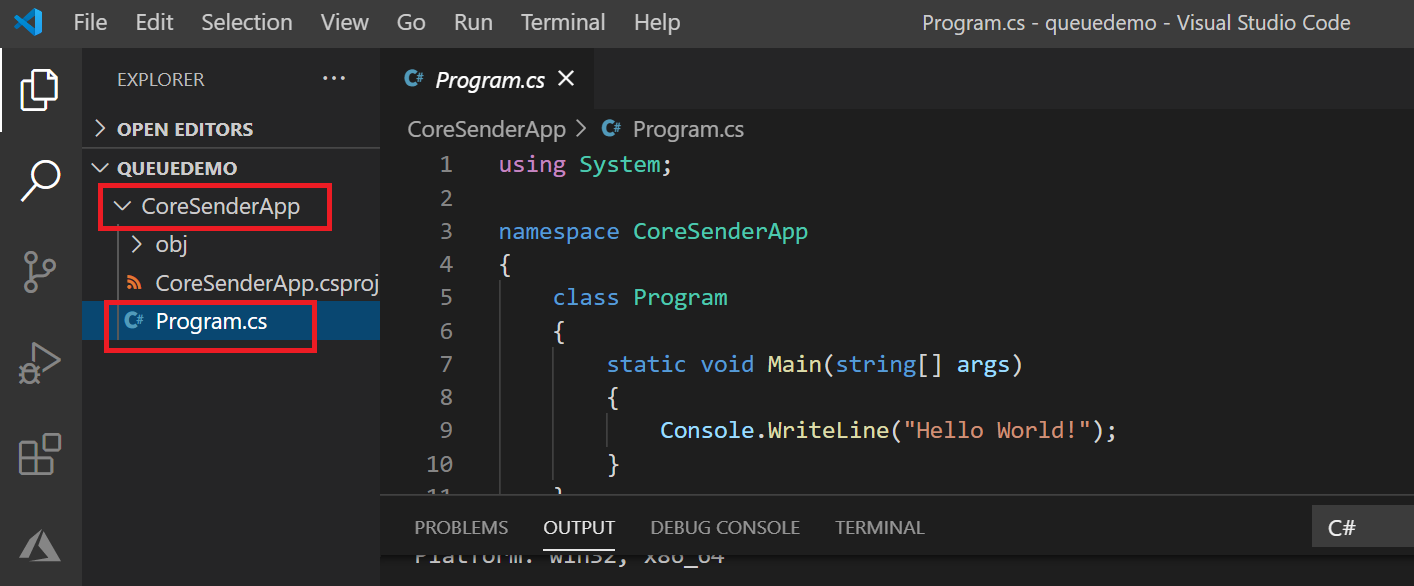
1. Enter the below command to open the CoreSenderApp project in VS Code

**cd CoreSenderApp**

**Code .**

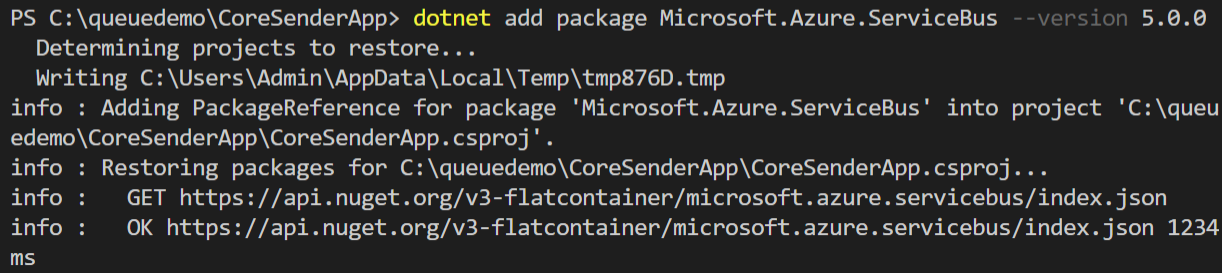


1. Now you will see the Visual Studio Code get launched with project

****

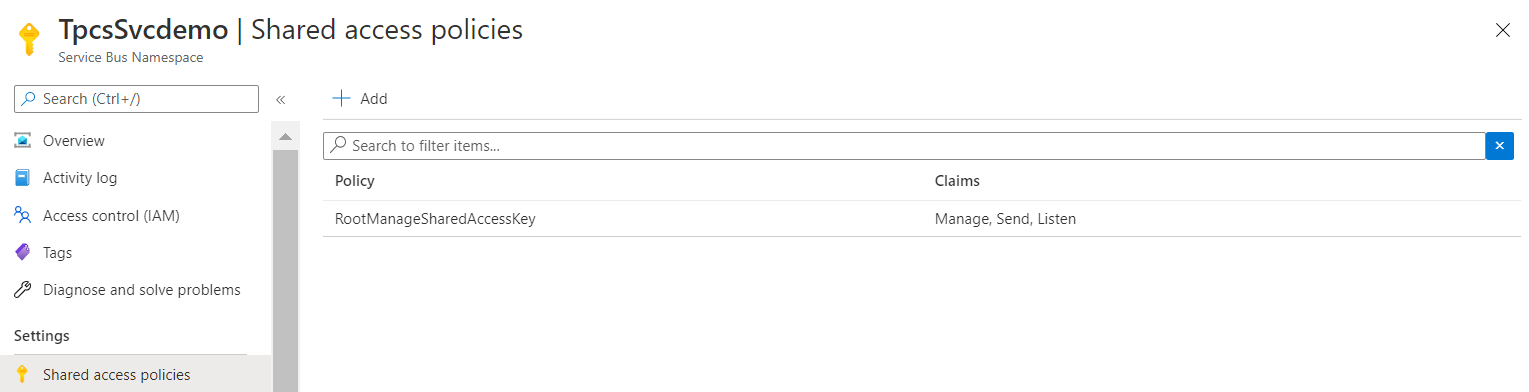
1. Now you add the Add the **Service Bus NuGet package.** Open the Terminal and type below command

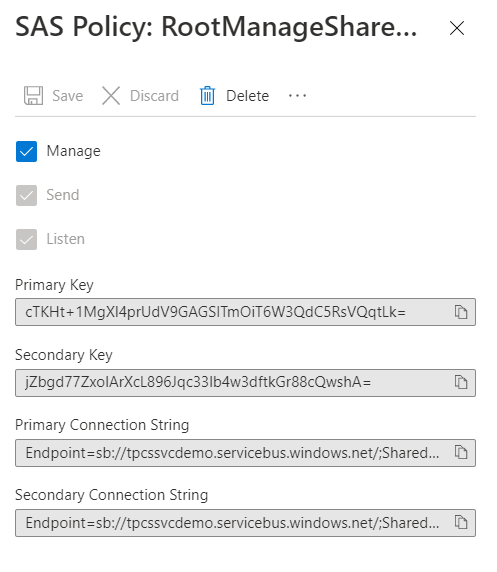
**dotnet add package Microsoft.Azure.ServiceBus --version 5.0.0**



## **Create Connmection Stric**

In Program.cs, add the following using statements at the top of the namespace definition, before the class declaration





# Write code to send messages to the queue

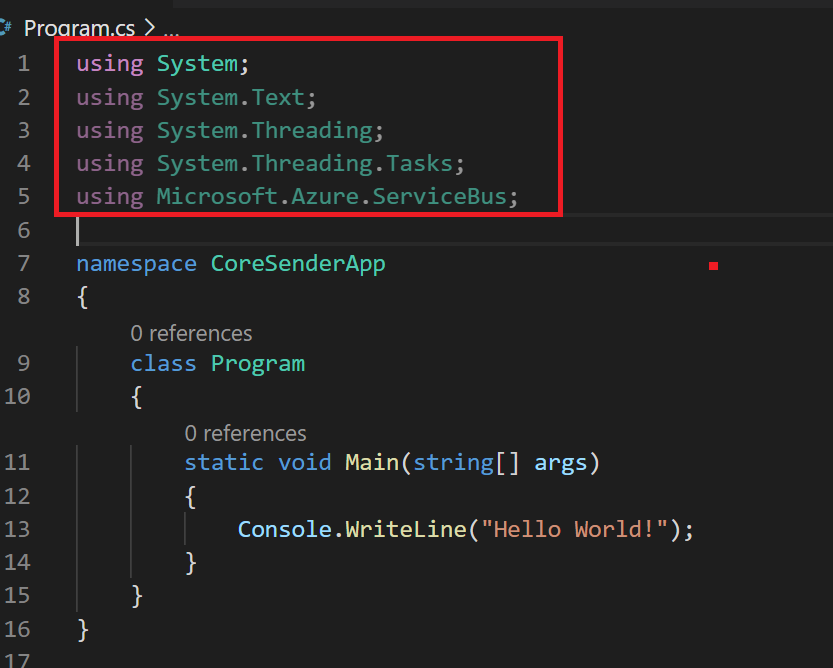
1. In Program.cs, add the following using statements at the top of the namespace definition, before the class declaration:

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using Microsoft.Azure.ServiceBus;

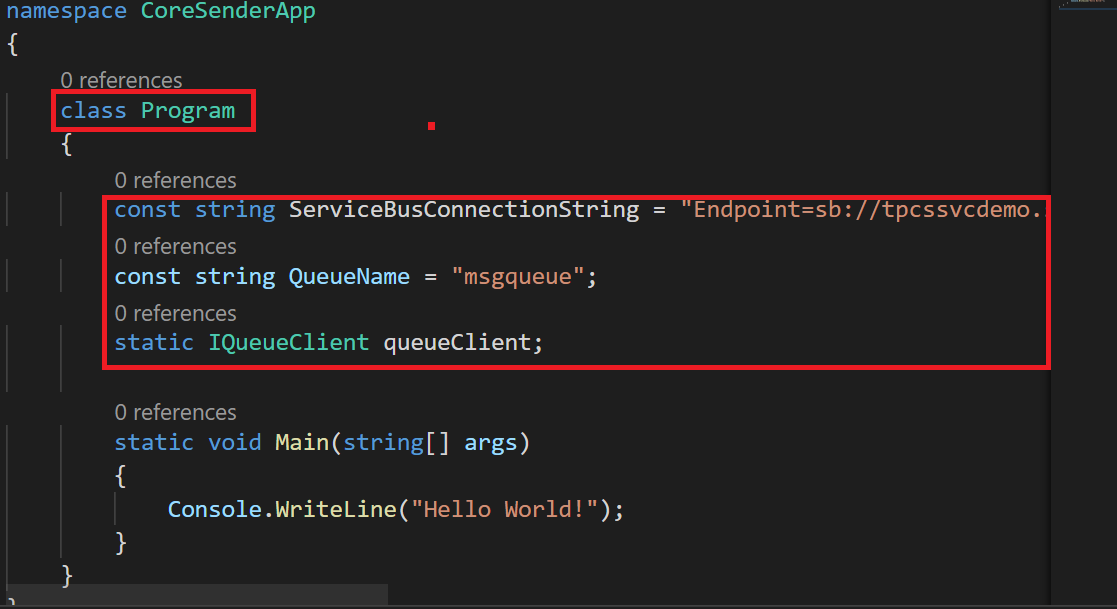


1. In  the Program class, declare the following variables:

const string ServiceBusConnectionString = "Endpoint=sb://tpcssvcdemo.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=cTKHt+1MgXl4prUdV9GAGSITmOiT6W3QdC5RsVQqtLk="

const string QueueName = “msgqueue“;

static IQueueClient queueClient;



1. Replace the **Main()** method with the following **async** Main method. It calls the **SendMessagesAsync()** method that you will add in the next step to send messages to the queue.

public static async Task Main(string[] args)

{

const int numberOfMessages = 10;

queueClient = new QueueClient(ServiceBusConnectionString, QueueName);

Console.WriteLine("======================================================");

Console.WriteLine("Press ENTER key to exit after sending all the messages.");

Console.WriteLine("======================================================");

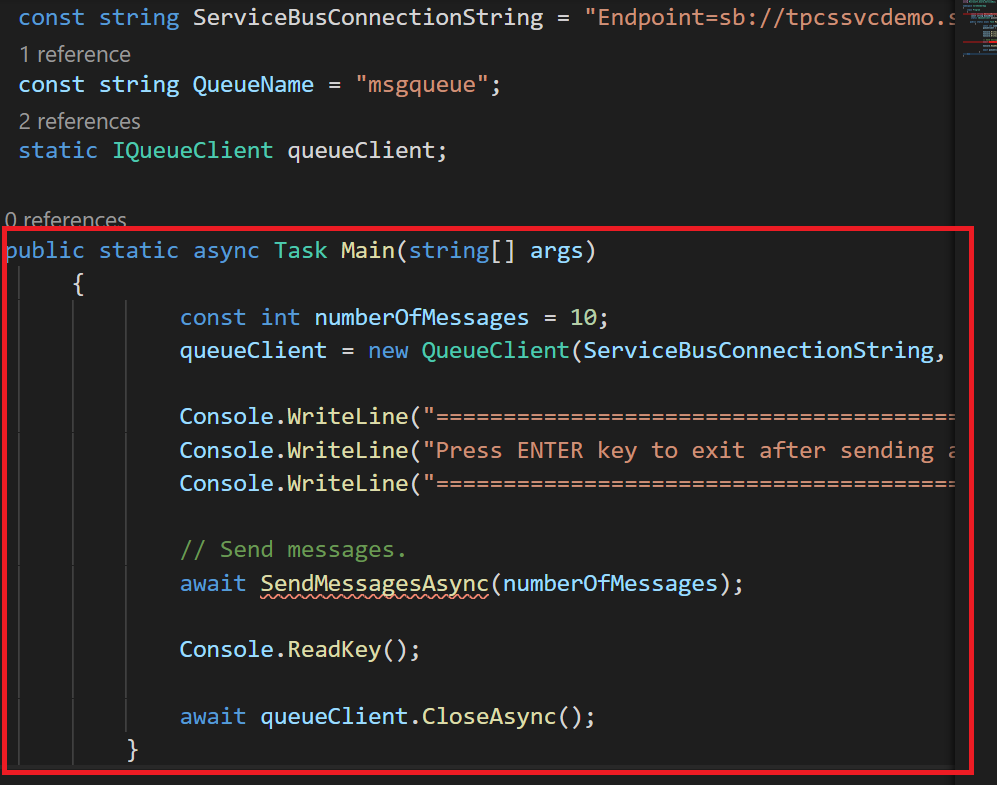
// Send messages.

await SendMessagesAsync(numberOfMessages);

Console.ReadKey();

await queueClient.CloseAsync();

}



1. Directly after the **MainAsync()** method, add the following **SendMessagesAsync()** method that does the work of sending the number of messages specified **MessagesToSend**(currently set to 10):

static async Task SendMessagesAsync(int numberOfMessagesToSend)

{

try

{

for (var i = 0; i < numberOfMessagesToSend; i++)

{

// Create a new message to send to the queue.

string messageBody = $"Message {i}";

var message = new Message(Encoding.UTF8.GetBytes(messageBody));

// Write the body of the message to the console.

Console.WriteLine($"Sending message: {messageBody}");

// Send the message to the queue.

await queueClient.SendAsync(message);

}

}

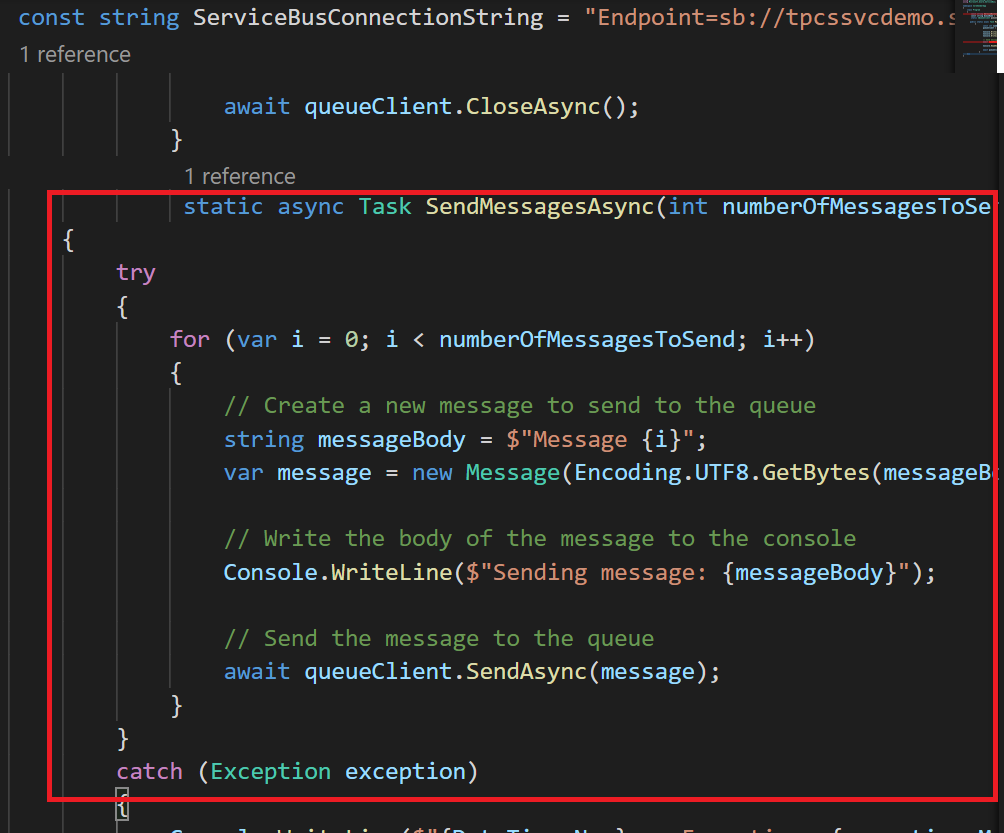
catch (Exception exception)

{

Console.WriteLine($"{DateTime.Now} :: Exception: {exception.Message}");

}

}

c

1. The Complete Program will look like this

namespace CoreSenderApp

{

using System;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using Microsoft.Azure.ServiceBus;

class Program

{

// Connection String for the namespace can be obtained from the Azure portal under the

// 'Shared Access policies' section.

const string ServiceBusConnectionString = "<your\_connection\_string>";

const string QueueName = "<your\_queue\_name>";

static IQueueClient queueClient;

public static async Task Main(string[] args)

{

const int numberOfMessages = 10;

queueClient = new QueueClient(ServiceBusConnectionString, QueueName);

Console.WriteLine("======================================================");

Console.WriteLine("Press ENTER key to exit after sending all the messages.");

Console.WriteLine("======================================================");

// Send messages.

await SendMessagesAsync(numberOfMessages);

Console.ReadKey();

await queueClient.CloseAsync();

}

static async Task SendMessagesAsync(int numberOfMessagesToSend)

{

try

{

for (var i = 0; i < numberOfMessagesToSend; i++)

{

// Create a new message to send to the queue

string messageBody = $"Message {i}";

var message = new Message(Encoding.UTF8.GetBytes(messageBody));

// Write the body of the message to the console

Console.WriteLine($"Sending message: {messageBody}");

// Send the message to the queue

await queueClient.SendAsync(message);

}

}

catch (Exception exception)

{

Console.WriteLine($"{DateTime.Now} :: Exception: {exception.Message}");

}

}

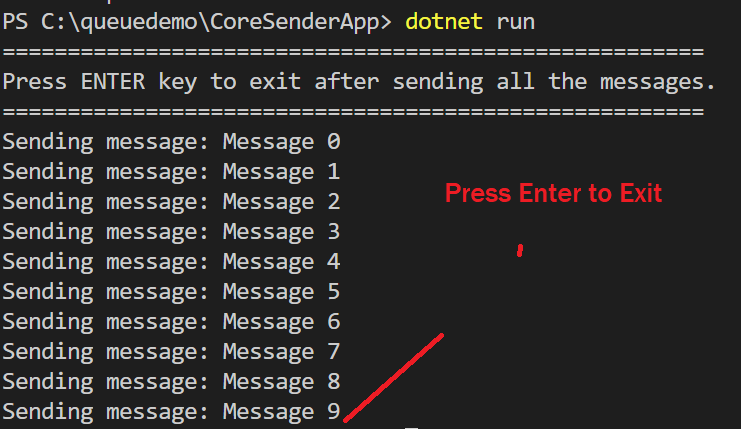
}

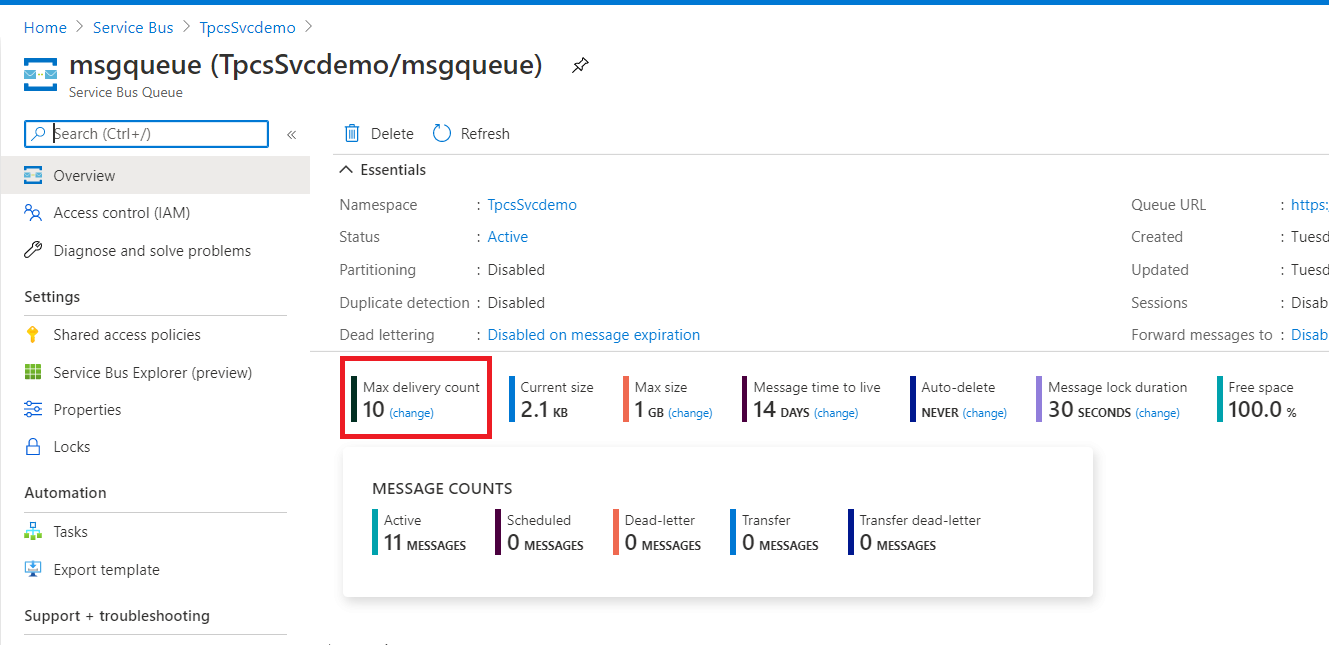
}

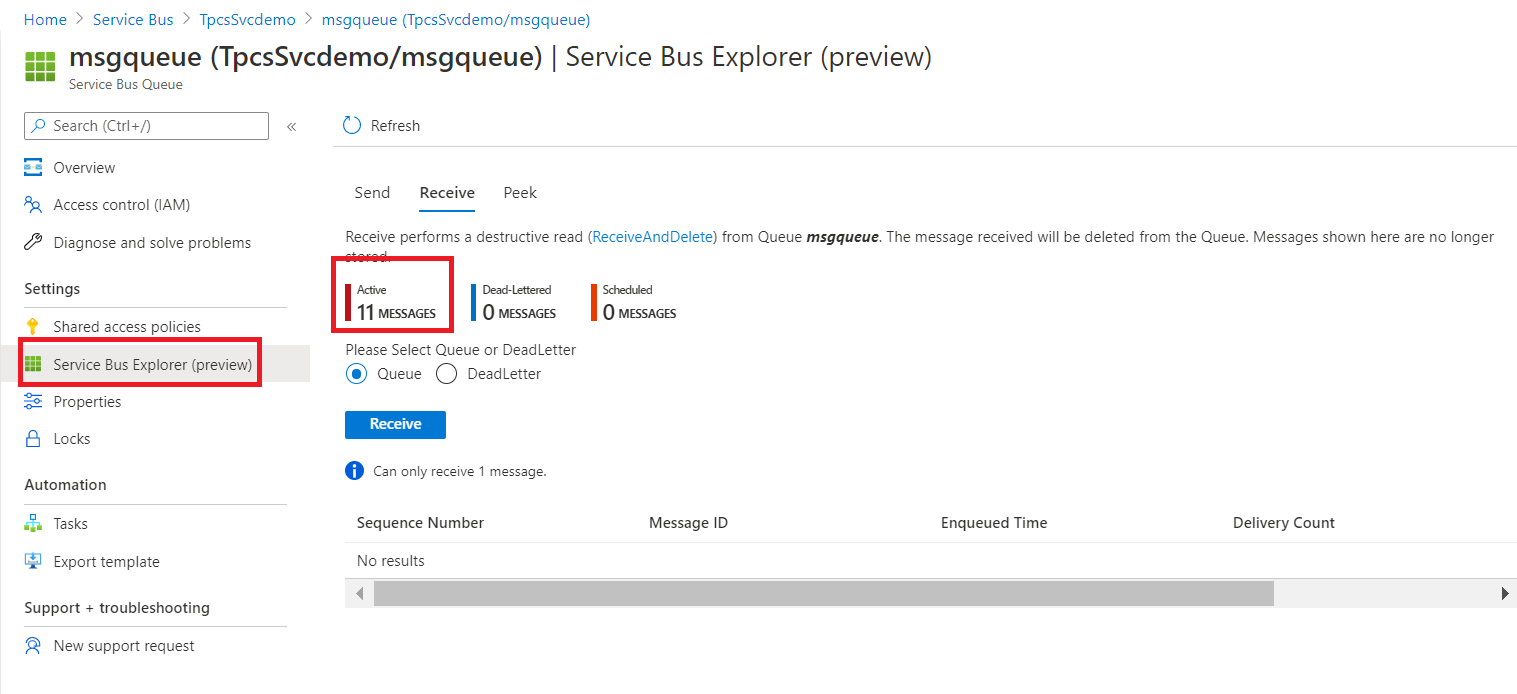
1. Save and Run the program and check the Azure portal.

**Dotnet Build**

**Dotnet Run**







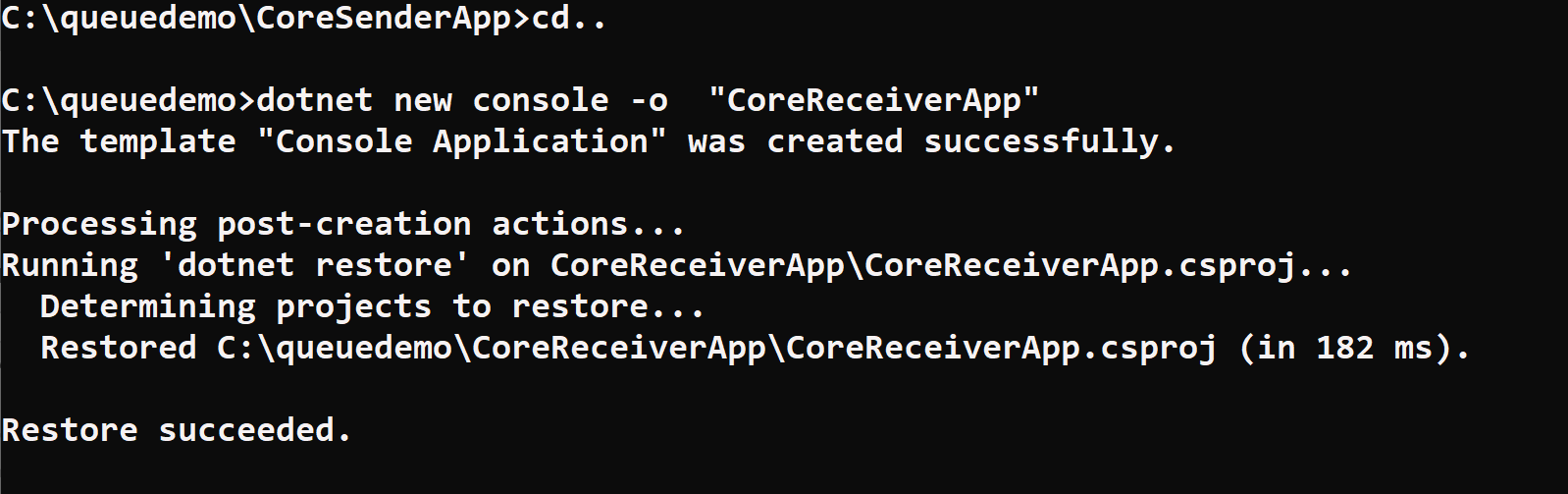
The **Active message count** value for the queue is now **10**. Each time you run this sender app without retrieving the messages, this value increases by 10.

# Receive messages from the queue

1. create another  create a **new Console App (.NET Core)**project for C#. This example names the app CoreSenderApp.

**cd ..**

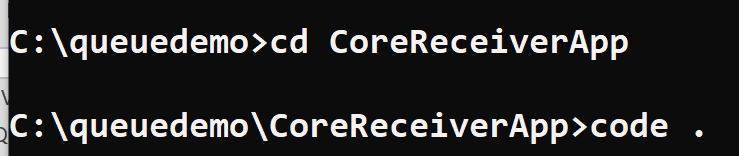
**dotnet new console -o “CoreReceiverApp“**



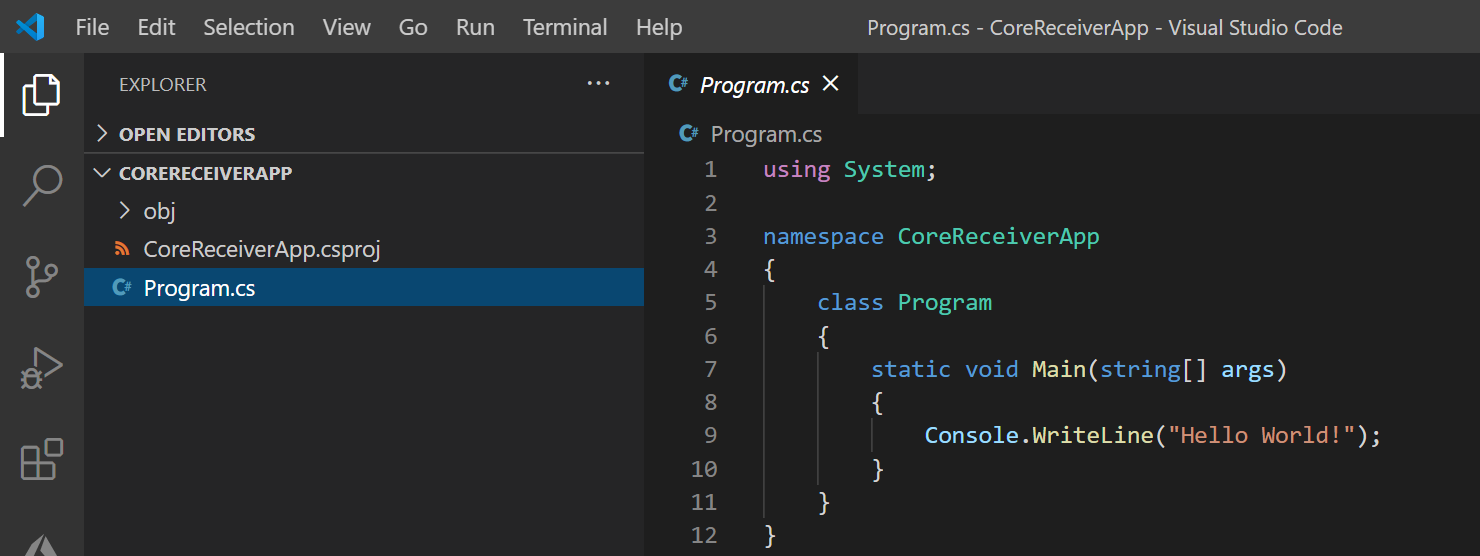
1. Enter the below command to open the CoreSenderApp project in VS Code

**cd CoreReceiverApp**

**Code .**

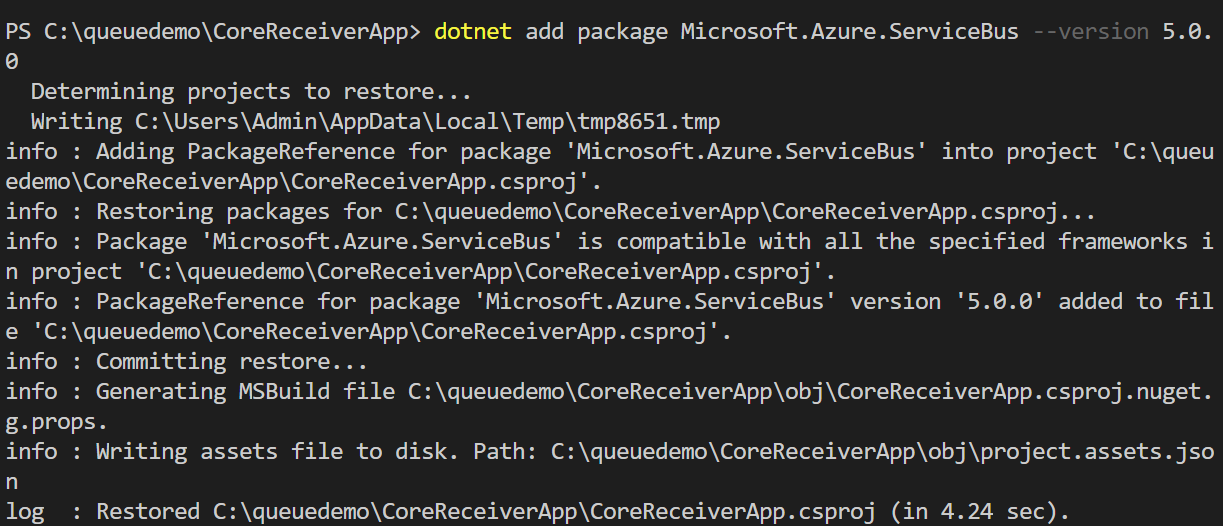


1. Now you will see the Visual Studio Code get launched with project



1. application. Install the **Microsoft.Azure.ServiceBus** **NuGet package,** as you did for the sender application.

**dotnet add package Microsoft.Azure.ServiceBus --version 5.0.0**



## Write code to receive messages from the queue

1. Replace the Code in **program.cs** and also replace the **connection string**

namespace CoreReceiverApp

{

using System;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using Microsoft.Azure.ServiceBus;

class Program

{

const string ServiceBusConnectionString = "<your\_connection\_string>";

const string QueueName = "<your\_queue\_name>";

static IQueueClient queueClient;

static void Main(string[] args)

{

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

}

static async Task MainAsync()

{

queueClient = new QueueClient(ServiceBusConnectionString, QueueName);

Console.WriteLine("======================================================");

Console.WriteLine("Press ENTER key to exit after receiving all the messages.");

Console.WriteLine("======================================================");

**// Register QueueClient's MessageHandler and receive messages in a loop**

RegisterOnMessageHandlerAndReceiveMessages();

Console.ReadKey();

await queueClient.CloseAsync();

}

static void RegisterOnMessageHandlerAndReceiveMessages()

{

var messageHandlerOptions = new MessageHandlerOptions(ExceptionReceivedHandler)

{

MaxConcurrentCalls = 1,

AutoComplete = false

};

queueClient.RegisterMessageHandler(ProcessMessagesAsync, messageHandlerOptions);

}

static async Task ProcessMessagesAsync(Message message, CancellationToken token)

{

Console.WriteLine($"Received message: SequenceNumber:{message.SystemProperties.SequenceNumber} Body:{Encoding.UTF8.GetString(message.Body)}");

await queueClient.CompleteAsync(message.SystemProperties.LockToken);

}

static Task ExceptionReceivedHandler(ExceptionReceivedEventArgs exceptionReceivedEventArgs)

{

Console.WriteLine($"Message handler encountered an exception {exceptionReceivedEventArgs.Exception}.");

var context = exceptionReceivedEventArgs.ExceptionReceivedContext;

Console.WriteLine("Exception context for troubleshooting:");

Console.WriteLine($"- Endpoint: {context.Endpoint}");

Console.WriteLine($"- Entity Path: {context.EntityPath}");

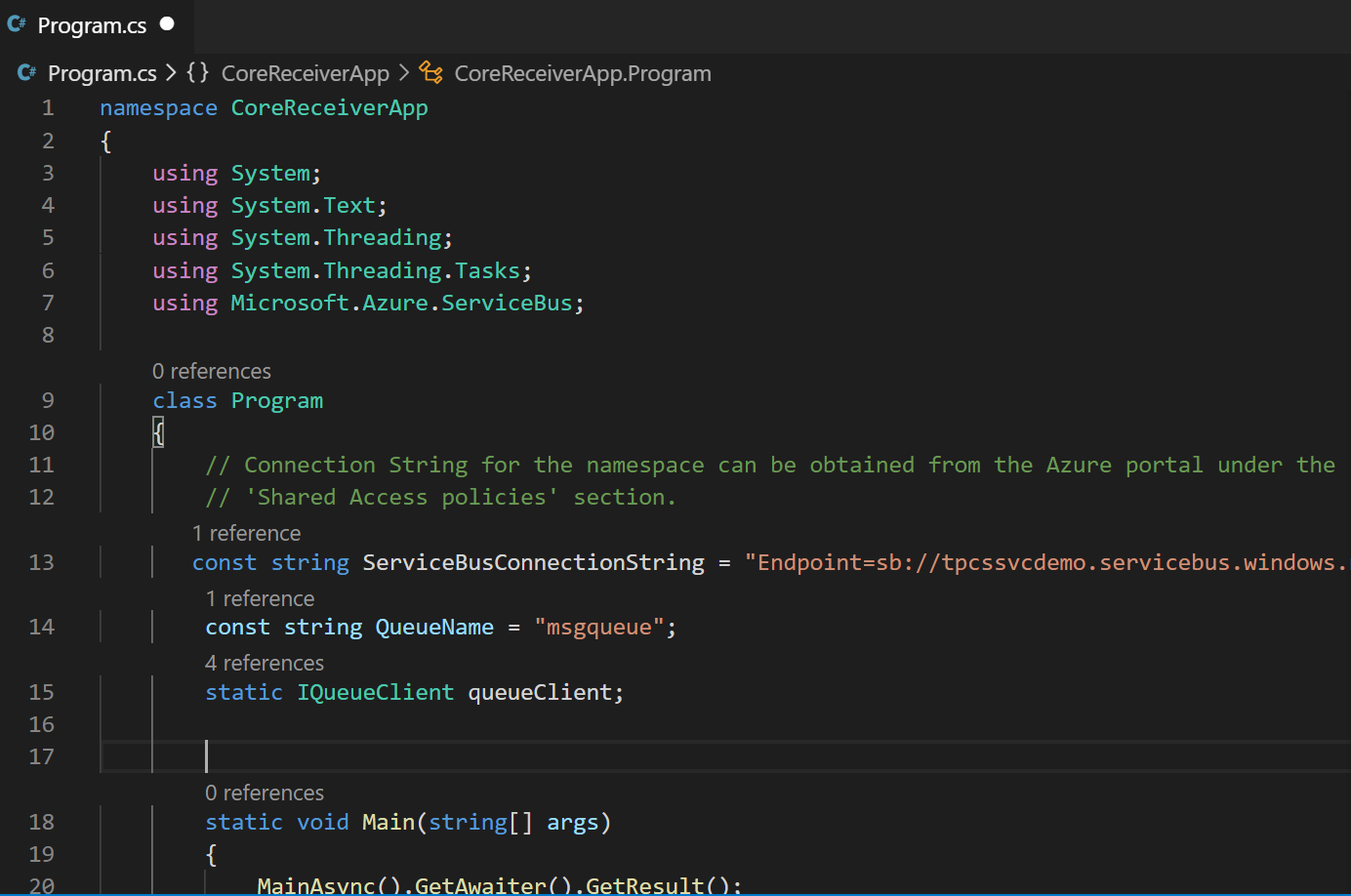
Console.WriteLine($"- Executing Action: {context.Action}");

return Task.CompletedTask;

}

}

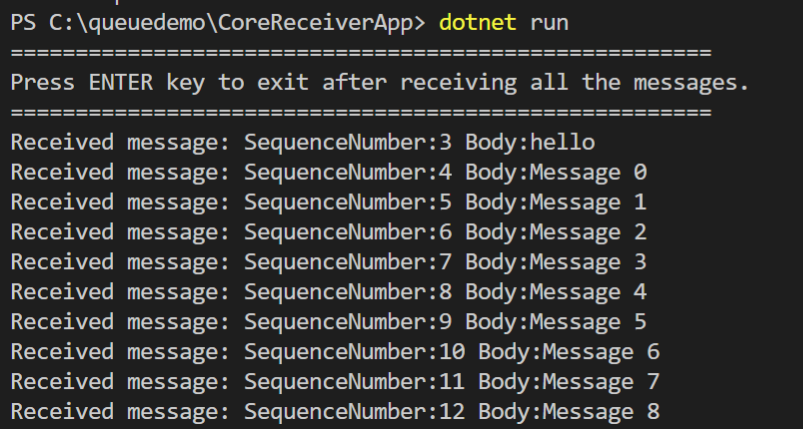
}

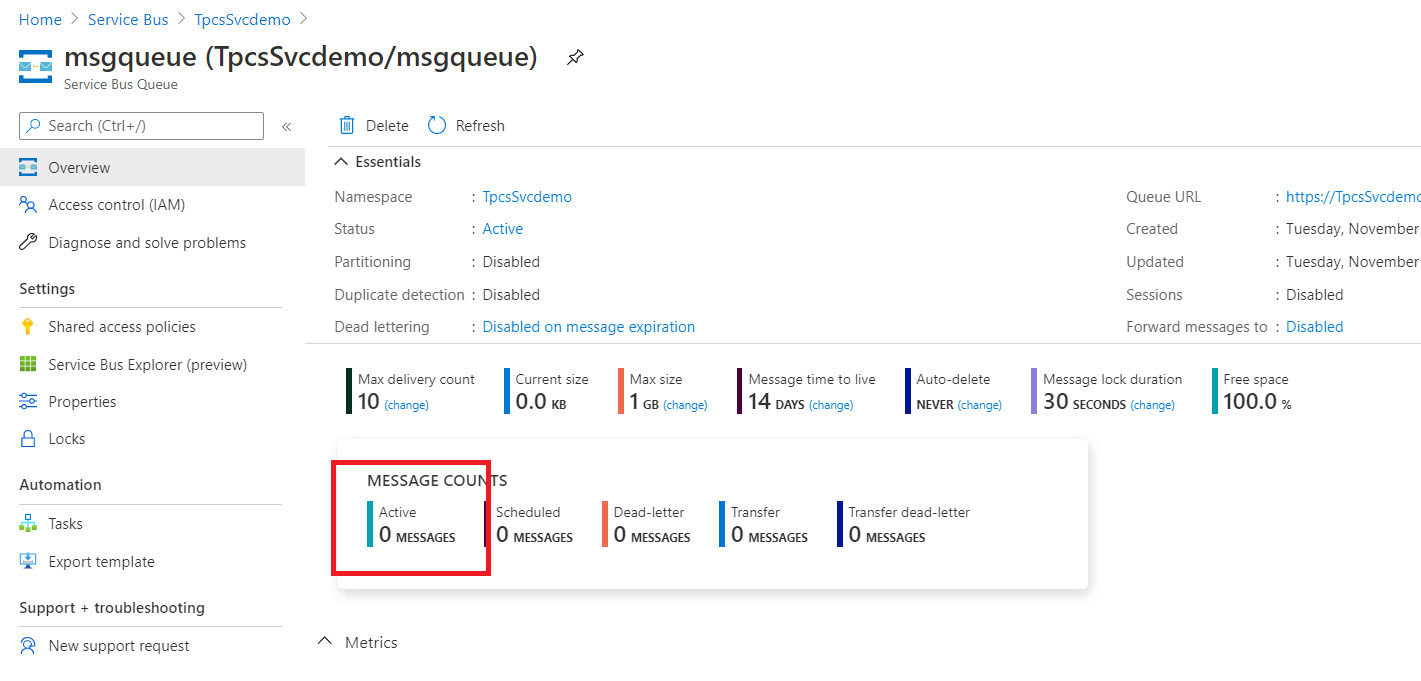


Save and Run the program and check the Azure portal The **Active message count** and **CURRENT** values are now **0**.

**Dotnet Build**

**Dotnet Run**





Congratulations! You've now created a queue, sent a set of messages to that queue, and received those messages from the same queue.

For More Reference : https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-get-started-with-queues