**Send and receive message from a Service Bus queue by using .NET.**

In this demo you will learn how to:

* Create a Service Bus namespace, and queue, using the Azure CLI.
* Create a .NET Core console application to send a set of messages to the queue.
* Create a .NET Core console application to receive those messages from the queue.

**Prerequisites**

* An **Azure account** with an active subscription. If you don't already have one, you can sign up for a free trial at <https://azure.com/free>.
* [Visual Studio Code](https://code.visualstudio.com/) on one of the [supported platforms](https://code.visualstudio.com/docs/supporting/requirements#_platforms).
* The [C# extension](https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp) for Visual Studio Code.

**Login to Azure**

In this section you will open your terminal and create some variables that will be used throughout the rest of the exercise to make command entry, and unique resource name creation, a bit easier.

1. Launch the [Azure Cloud Shell](https://shell.azure.com/) and select **Bash** and the environment.
2. Create variables used in the Azure CLI commands. Replace <myLocation> with a region near you.

myLocation="southeastasia"

myNameSpaceName=az204svcbus$RANDOM

myGroup="az204-svcbus-rg"

**Create Azure resources**

1. Create a resource group to hold the Azure resources you will be creating.

az group create --name $myGroup --location $myLocation

1. Create a Service Bus messaging namespace. The command below will create a namespace using the variable you created earlier. The operation will take a few minutes to complete.

az servicebus namespace create --resource-group $myGroup --name $myNameSpaceName --location $myLocation

1. Create a Service Bus queue

az servicebus queue create --resource-group $myGroup --namespace-name $myNameSpaceName --name az204-queue

**Retrieve the connection string for the Service Bus Namespace**

1. Open the Azure portal and navigate to the **az204-svcbus-rg** resource group.
2. Select the **az204svcbus** resource you just created.
3. Select **Shared access policies** in the **Settings** section, then select the **RootManageSharedAccessKey** policy.
4. Copy the **Primary Connection String** from the dialog box that opens up and save it to a file, or leave the portal open and copy the key when needed.

**Create console app to send messages to the queue**

1. Open a local terminal and create, and change in to, a directory named *az204svcbus* and then run the command to launch Visual Studio Code.

code .

1. Open the terminal in VS Code by selecting **Terminal > New Terminal** in the menu bar and run the following commands to create the console app and add the **Azure.Messaging.ServiceBus** package.

dotnet new console -f netcoreapp3.1

dotnet add package Azure.Messaging.ServiceBus

1. In *Program.cs*, add the following using statements at the top of the file after the current using statement.

using System.Threading.Tasks;

using Azure.Messaging.ServiceBus;

1. In the Program class, add the following two static properties. Set the ServiceBusConnectionString variable to the connection string that you obtained earlier.

// connection string to your Service Bus namespace

static string connectionString = "<NAMESPACE CONNECTION STRING>";

// name of your Service Bus topic

static string queueName = "az204-queue";

1. Declare the following static properties in the Program class. See code comments for details.

// the client that owns the connection and can be used to create senders and receivers

static ServiceBusClient client;

// the sender used to publish messages to the queue

static ServiceBusSender sender;

// number of messages to be sent to the queue

private const int numOfMessages = 3;

1. Replace the Main() method with the following **async**Main method.

static async Task Main()

 {

 // Create the clients that we'll use for sending and processing messages.

 client = new ServiceBusClient(connectionString);

 sender = client.CreateSender(queueName);

 // create a batch

 using ServiceBusMessageBatch messageBatch = await sender.CreateMessageBatchAsync();

 for (int i = 1; i <= 3; i++)

 {

 // try adding a message to the batch

 if (!messageBatch.TryAddMessage(new ServiceBusMessage($"Message {i}")))

 {

 // if an exception occurs

 throw new Exception($"Exception {i} has occurred.");

 }

 }

 try

 {

 // Use the producer client to send the batch of messages to the Service Bus queue

 await sender.SendMessagesAsync(messageBatch);

 Console.WriteLine($"A batch of {numOfMessages} messages has been published to the queue.");

 }

 finally

 {

 // Calling DisposeAsync on client types is required to ensure that network

 // resources and other unmanaged objects are properly cleaned up.

 await sender.DisposeAsync();

 await client.DisposeAsync();

 }

 Console.WriteLine("Press any key to end the application");

 Console.ReadKey();

 }

1. Save the file and run the dotnet build command to ensure there are no errors.
2. Run the program and wait for the confirmation message.

A batch of 3 messages has been published to the queue.

1. Login to the Azure portal and navigate to your Service Bus namespace. On the Overview page, select the *az204-queue* queue in the bottom-middle pane.

Notice the following values in the Essentials section:

* + The **Active** message count value for the queue is now **3**. Each time you run this sender app without retrieving the messages, this value increases by 3.
	+ The **current size** of the queue increments each time the app adds messages to the queue.
	+ In the **Messages** chart in the bottom **Metrics** section, you can see that there are three incoming messages for the queue.

**Update project to receive messages to the queue**

In this section you'll modify the program to receive messages from the queue.

1. In the Program class, delete the static properties that follow ServiceBusClient. We'll keep using connectionString, queueName, and ServiceBusClient for the rest of the exercise. Add the following after the ServiceBusClient static property.

// the processor that reads and processes messages from the queue

static ServiceBusProcessor processor;

1. Add the following methods to the Program class to handle messages and any errors.

// handle received messages

static async Task MessageHandler(ProcessMessageEventArgs args)

{

 string body = args.Message.Body.ToString();

 Console.WriteLine($"Received: {body}");

 // complete the message. messages is deleted from the queue.

 await args.CompleteMessageAsync(args.Message);

}

// handle any errors when receiving messages

static Task ErrorHandler(ProcessErrorEventArgs args)

{

 Console.WriteLine(args.Exception.ToString());

 return Task.CompletedTask;

}

1. Replace the Main() method. It calls the ReceiveMessages method to receive messages from the queue.

static async Task Main()

{

 // Create the client object that will be used to create sender and receiver objects

 client = new ServiceBusClient(connectionString);

 // create a processor that we can use to process the messages

 processor = client.CreateProcessor(queueName, new ServiceBusProcessorOptions());

 try

 {

 // add handler to process messages

 processor.ProcessMessageAsync += MessageHandler;

 // add handler to process any errors

 processor.ProcessErrorAsync += ErrorHandler;

 // start processing

 await processor.StartProcessingAsync();

 Console.WriteLine("Wait for a minute and then press any key to end the processing");

 Console.ReadKey();

 // stop processing

 Console.WriteLine("\nStopping the receiver...");

 await processor.StopProcessingAsync();

 Console.WriteLine("Stopped receiving messages");

 }

 finally

 {

 // Calling DisposeAsync on client types is required to ensure that network

 // resources and other unmanaged objects are properly cleaned up.

 await processor.DisposeAsync();

 await client.DisposeAsync();

 }

}

1. Use the dotnet build command to ensure there are no errors.
2. Use the dotnet run command to run the application. You should see the received messages. Press any key to stop the receiver and the application.

Wait for a minute and then press any key to end the processing

Received: Message 1

Received: Message 2

Received: Message 3

Stopping the receiver...

Stopped receiving messages

1. Check the portal again. Notice that the **Active Message Count** value is now 0. You may need to refresh the portal page.

**Clean up resources**

When the resources are no longer needed, you can use the az group delete command in the Azure Cloud Shell to remove the resource group.

az group delete --name $myGroup --no-wait