Quickstart: Send and receive messages from an Azure Service Bus queue (.NET)

Article • 12/05/2023

In this quickstart, you'll do the following steps:

1. Create a Service Bus namespace, using the Azure portal.
2. Create a Service Bus queue, using the Azure portal.
3. Write a .NET console application to send a set of messages to the queue.
4. Write a .NET console application to receive those messages from the queue.

> ⓘ **Note**
>
> This quick start provides step-by-step instructions to implement a simple scenario of sending a batch of messages to a Service Bus queue and then receiving them. For an overview of the .NET client library, see Azure Service Bus client library for .NET. For more samples, see Service Bus .NET samples on GitHub.

Prerequisites

If you're new to the service, see Service Bus overview before you do this quickstart.

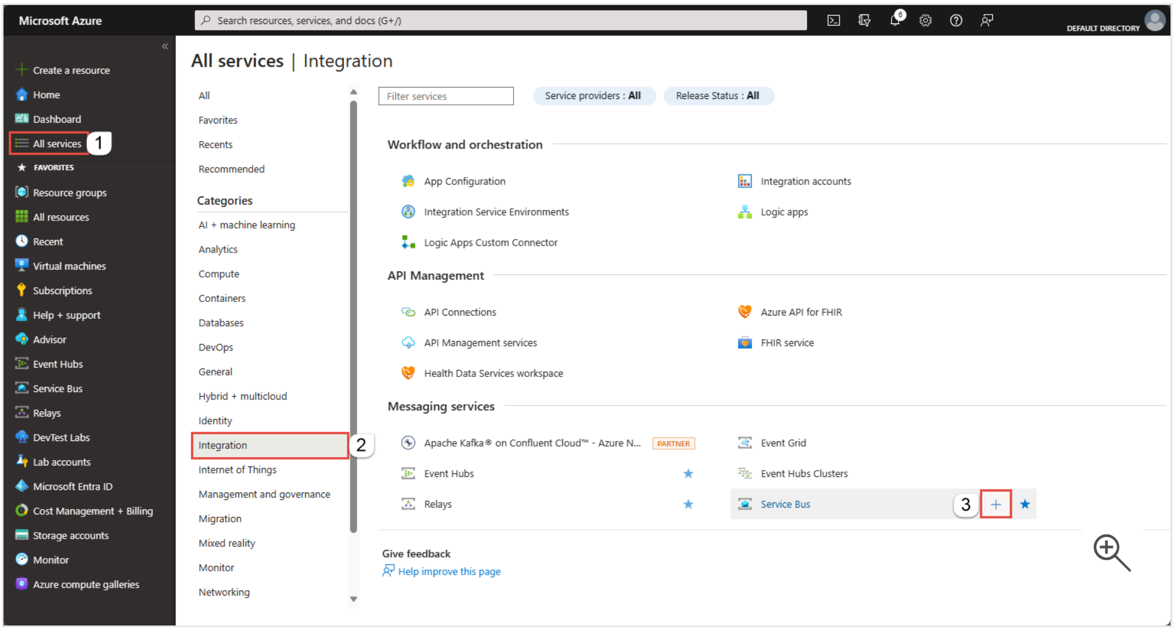
- **Azure subscription**. To use Azure services, including Azure Service Bus, you need a subscription. If you don't have an existing Azure account, you can sign up for a free trial.
- **Visual Studio 2022**. The sample application makes use of new features that were introduced in C# 10. You can still use the Service Bus client library with previous C# language versions, but the syntax might vary. To use the latest syntax, we recommend that you install .NET 6.0, or higher and set the language version to `latest`. If you're using Visual Studio, versions before Visual Studio 2022 aren't compatible with the tools needed to build C# 10 projects.

Create a namespace in the Azure portal

To begin using Service Bus messaging entities in Azure, you must first create a namespace with a name that is unique across Azure. A namespace provides a scoping container for Service Bus resources (queues, topics, etc.) within your application.

To create a namespace:

1. Sign in to the Azure portal .
2. Navigate to the **All services** page .
3. On the left navigation bar, select **Integration** from the list of categories, hover the mouse over **Service Bus**, and then select **+** button on the Service Bus tile.



4. In the **Basics** tag of the **Create namespace** page, follow these steps:

   a. For **Subscription**, choose an Azure subscription in which to create the namespace.

   b. For **Resource group**, choose an existing resource group in which the namespace will live, or create a new one.

   c. Enter a **name for the namespace**. The namespace name should adhere to the following naming conventions:

      - The name must be unique across Azure. The system immediately checks to see if the name is available.
      - The name length is at least 6 and at most 50 characters.
      - The name can contain only letters, numbers, hyphens "-".
      - The name must start with a letter and end with a letter or number.
      - The name doesn't end with "-sb" or "-mgmt".

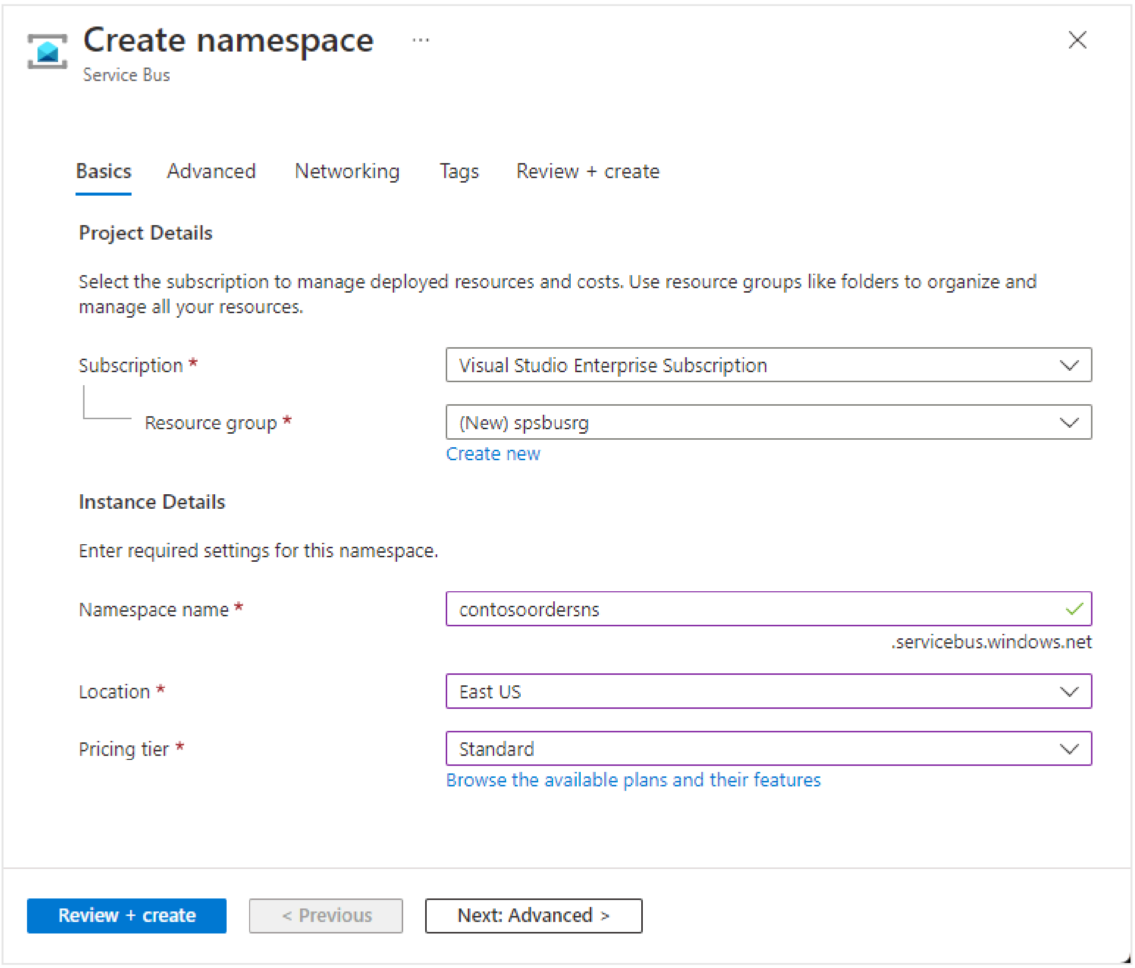
d. For **Location**, choose the region in which your namespace should be hosted.

e. For **Pricing tier**, select the pricing tier (Basic, Standard, or Premium) for the namespace. For this quickstart, select **Standard**.

> ⓘ **Important**
>
> If you want to use [topics and subscriptions](), choose either Standard or Premium. Topics/subscriptions aren't supported in the Basic pricing tier.

If you selected the **Premium** pricing tier, specify the number of **messaging units**. The premium tier provides resource isolation at the CPU and memory level so that each workload runs in isolation. This resource container is called a messaging unit. A premium namespace has at least one messaging unit. You can select 1, 2, 4, 8 or 16 messaging units for each Service Bus Premium namespace. For more information, see Service Bus Premium Messaging.

f. Select **Review + create** at the bottom of the page.



Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * : Visual Studio Enterprise Subscription

Resource group * : (New) spsbusrg

Create new

Instance Details

Enter required settings for this namespace.

Namespace name * : contosoordersns .servicebus.windows.net

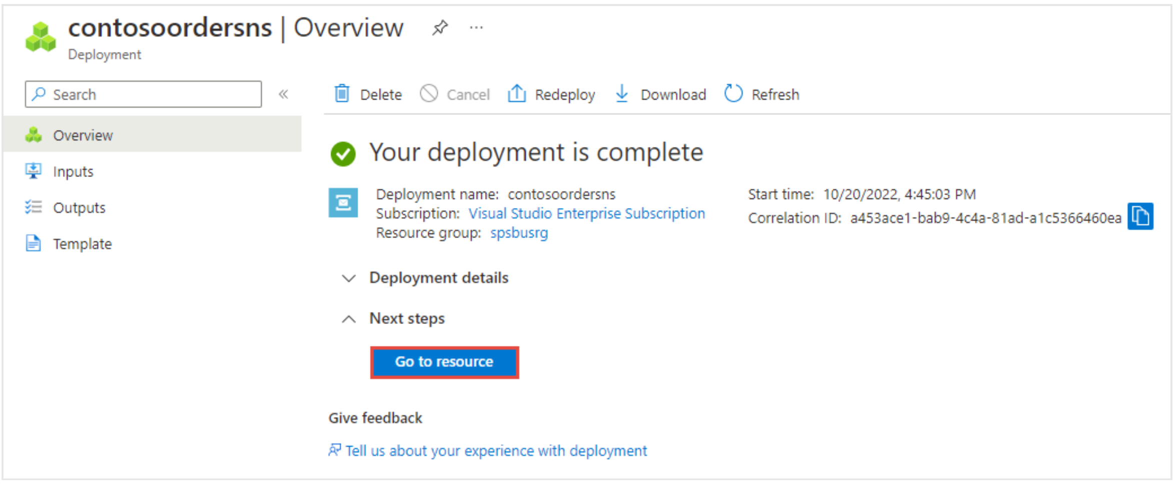
Location * : East US

Pricing tier * : Standard

Browse the available plans and their features

g. On the **Review + create** page, review settings, and select **Create**.

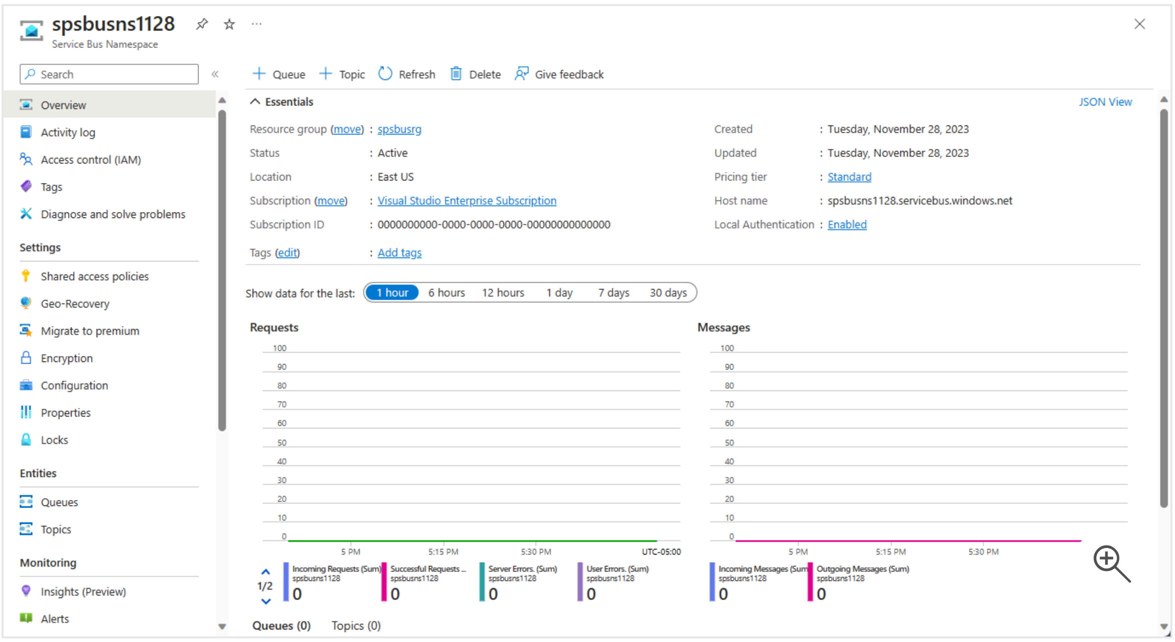
5. Once the deployment of the resource is successful, select **Go to resource** on the deployment page.



Your deployment is complete

Deployment name: contosoordersns
Subscription: Visual Studio Enterprise Subscription
Resource group: spsbusrg
Start time: 10/20/2022, 4:45:03 PM
Correlation ID: a453ace1-bab9-4c4a-81ad-a1c5366460ea

6. You see the home page for your service bus namespace.



Resource group (move)	: spsbusrg	Created	: Tuesday, November 28, 2023
Status	: Active	Updated	: Tuesday, November 28, 2023
Location	: East US	Pricing tier	: Standard
Subscription (move)	: Visual Studio Enterprise Subscription	Host name	: spsbusns1128.servicebus.windows.net
Subscription ID	: 0000000000-0000-0000-0000-0000000000000	Local Authentication	: Enabled
Tags (edit)	: Add tags		

Show data for the last: 1 hour 6 hours 12 hours 1 day 7 days 30 days

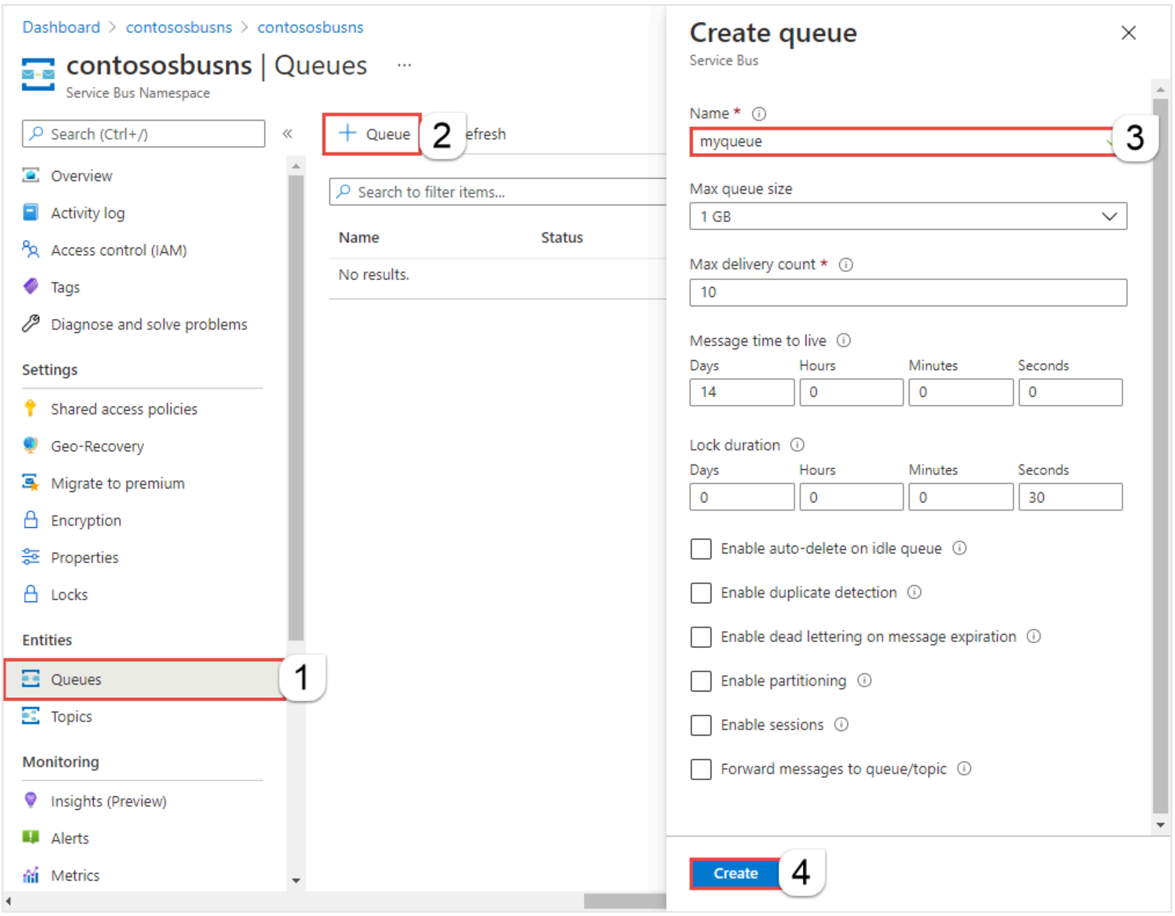
Create a queue in the Azure portal

1. On the **Service Bus Namespace** page, select **Queues** in the left navigational menu.

2. On the **Queues** page, select **+ Queue** on the toolbar.

3. Enter a **name** for the queue, and leave the other values with their defaults.

4. Now, select **Create**.



Create queue
Service Bus

Name *: myqueue
Max queue size: 1 GB
Max delivery count *: 10
Message time to live: Days 14, Hours 0, Minutes 0, Seconds 0
Lock duration: Days 0, Hours 0, Minutes 0, Seconds 30
- [ ] Enable auto-delete on idle queue
- [ ] Enable duplicate detection
- [ ] Enable dead lettering on message expiration
- [ ] Enable partitioning
- [ ] Enable sessions
- [ ] Forward messages to queue/topic

ⓘ **Important**

If you are new to Azure, you might find the **Connection String** option easier to follow. Select the **Connection String** tab to see instructions on using a connection string in this quickstart. We recommend that you use the **Passwordless** option in real-world applications and production environments.

Authenticate the app to Azure

This quick start shows you two ways of connecting to Azure Service Bus: **passwordless** and **connection string**.

The first option shows you how to use your security principal in Microsoft Entra ID and role-based access control (RBAC) to connect to a Service Bus namespace. You don't need to worry about having hard-coded connection string in your code or in a configuration file or in a secure storage like Azure Key Vault.

The second option shows you how to use a connection string to connect to a Service Bus namespace. If you are new to Azure, you may find the connection string option easier to follow. We recommend using the passwordless option in real-world applications and production environments. For more information, see Authentication and authorization. You can also read more about passwordless authentication on the overview page.

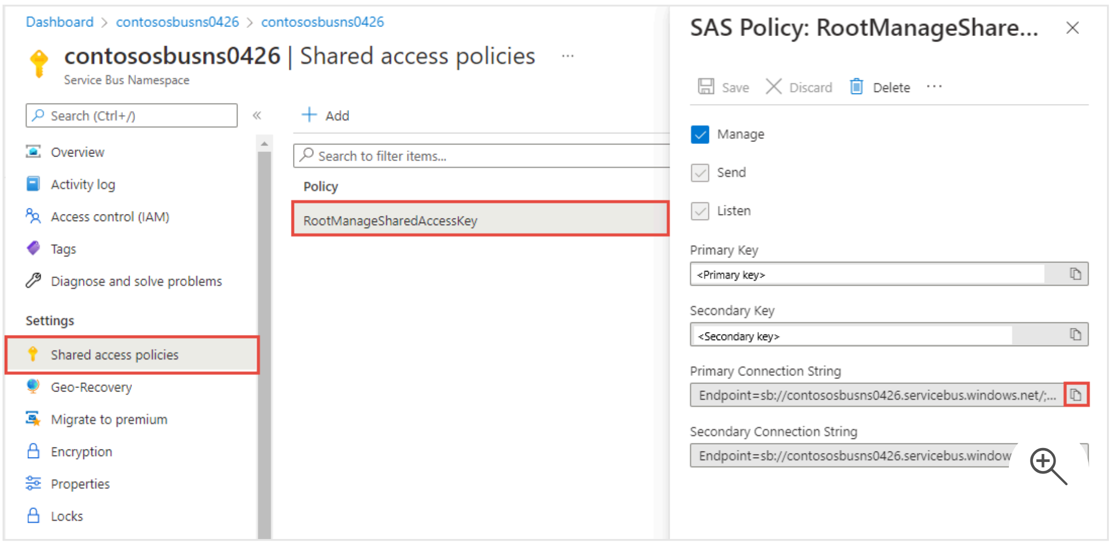
Connection String

Get the connection string

Creating a new namespace automatically generates an initial Shared Access Signature (SAS) policy with primary and secondary keys, and primary and secondary connection strings that each grant full control over all aspects of the namespace. See Service Bus authentication and authorization for information about how to create rules with more constrained rights for regular senders and receivers.

A client can use the connection string to connect to the Service Bus namespace. To copy the primary connection string for your namespace, follow these steps:

1. On the **Service Bus Namespace** page, select **Shared access policies** on the left menu.
2. On the **Shared access policies** page, select **RootManageSharedAccessKey**.
3. In the **Policy: RootManageSharedAccessKey** window, select the copy button next to **Primary Connection String**, to copy the connection string to your clipboard for later use. Paste this value into Notepad or some other temporary location.



Primary Key: <Primary key>
Secondary Key: <Secondary key>
Primary Connection String: Endpoint=sb://contososbusns0426.servicebus.windows.net/;...
Secondary Connection String: Endpoint=sb://contososbusns0426.servicebus.window

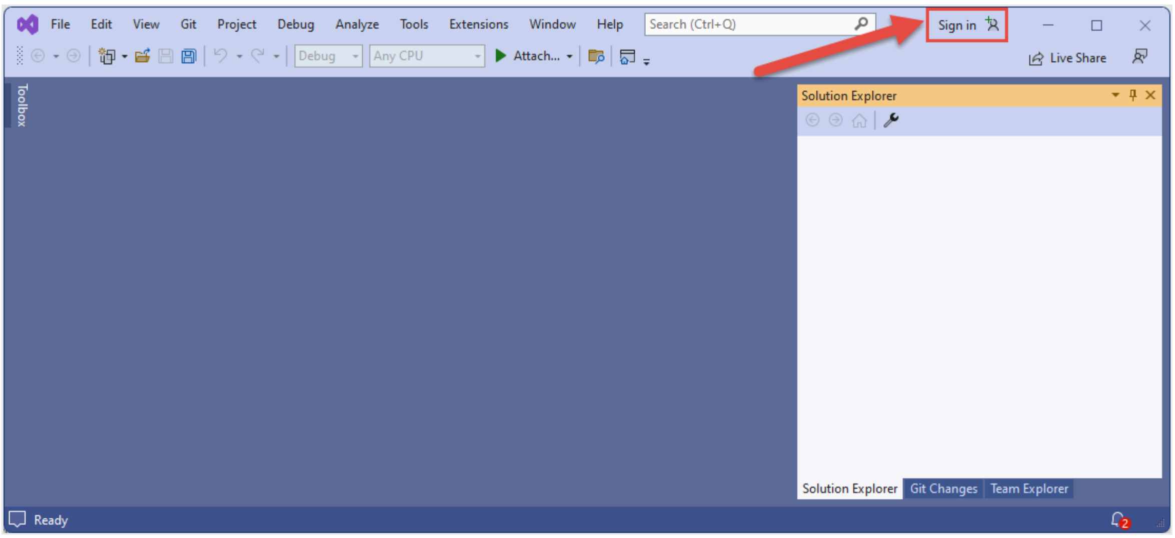
You can use this page to copy primary key, secondary key, primary connection string, and secondary connection string.

Launch Visual Studio and sign-in to Azure

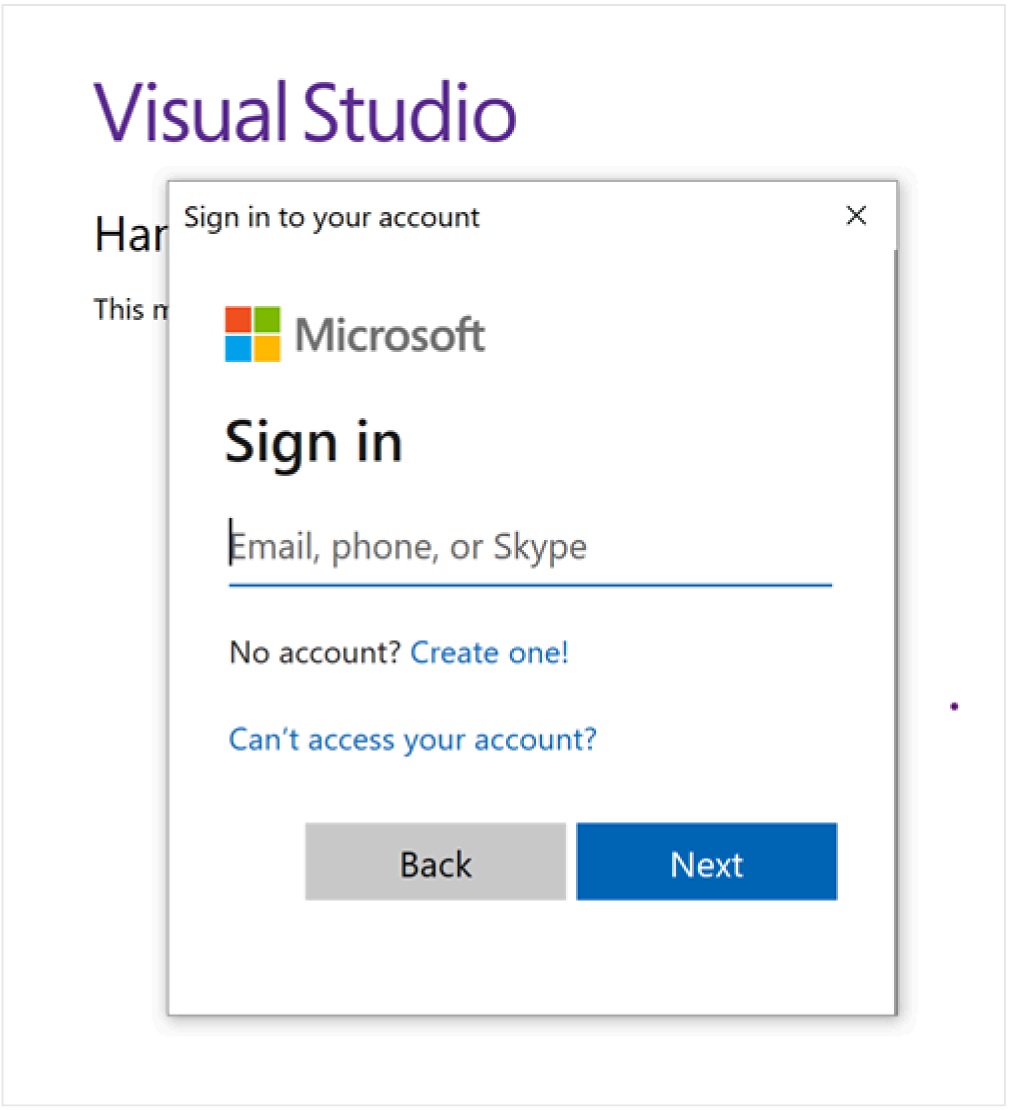
You can authorize access to the service bus namespace using the following steps:

1. Launch Visual Studio. If you see the **Get started** window, select the **Continue without code** link in the right pane.

2. Select the **Sign in** button in the top right of Visual Studio.



3. Sign-in using the Microsoft Entra account you assigned a role to previously.



Send messages to the queue

This section shows you how to create a .NET console application to send messages to a Service Bus queue.

ⓘ Note

This quick start provides step-by-step instructions to implement a simple scenario of sending a batch of messages to a Service Bus queue and then receiving them. For more samples on other and advanced scenarios, see Service Bus .NET samples on GitHub .

Create a console application

1. In Visual Studio, select **File** -> **New** -> **Project** menu.

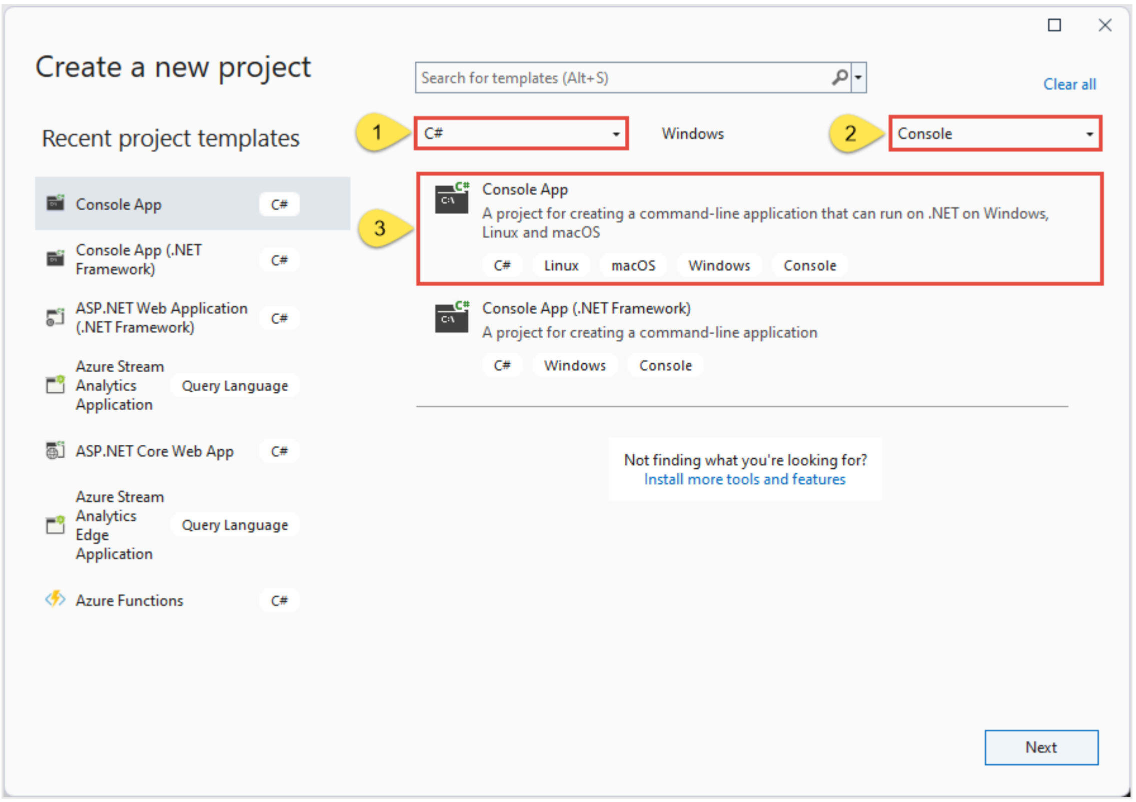
2. On the **Create a new project** dialog box, do the following steps: If you don't see this dialog box, select **File** on the menu, select **New**, and then select **Project**.

   a. Select **C#** for the programming language.

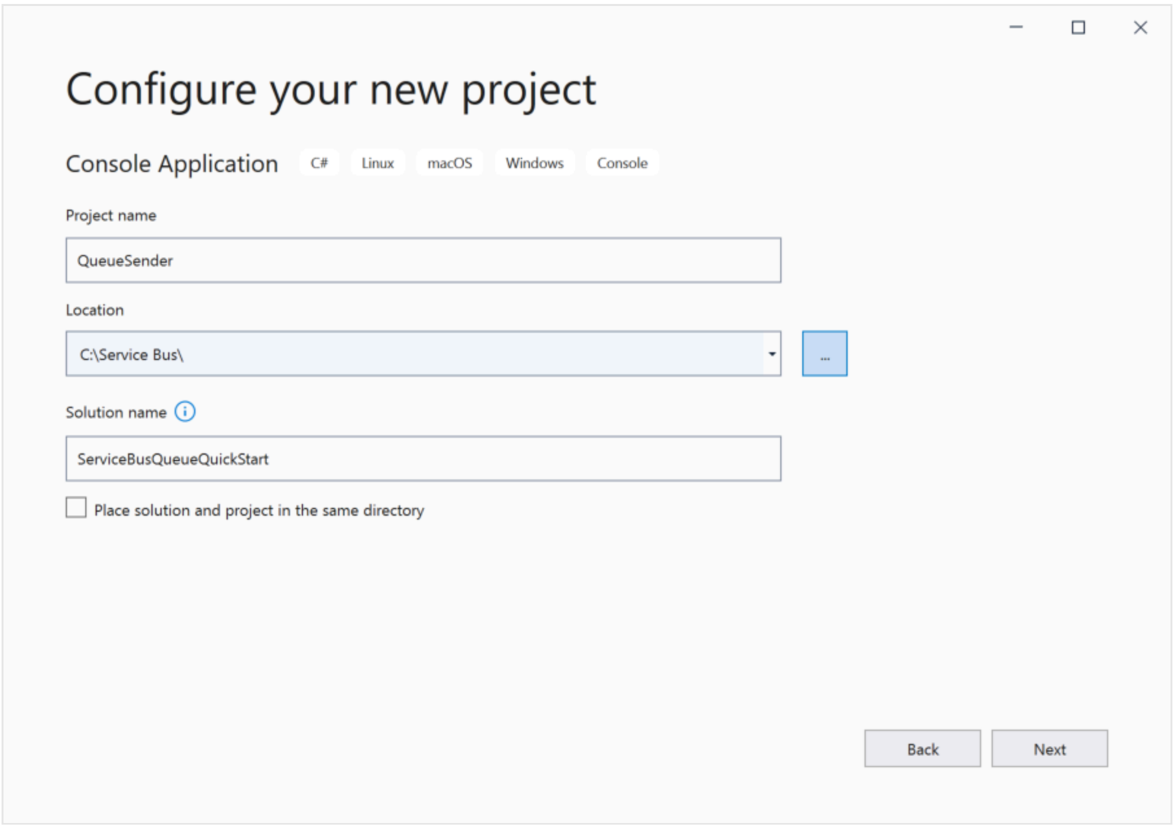
   b. Select **Console** for the type of the application.

   c. Select **Console App** from the results list.

   d. Then, select **Next**.


Create a new project
Search for templates (Alt+S)
Clear all
Recent project templates
1
C#
Windows
2
Console
3
Console App C#
Console App (.NET Framework) C#
ASP.NET Web Application (.NET Framework) C#
Azure Stream Analytics Application Query Language
ASP.NET Core Web App C#
Azure Stream Analytics Edge Application Query Language
Azure Functions C#
Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Console
Console App (.NET Framework)
A project for creating a command-line application
C# Windows Console
Not finding what you're looking for?
Install more tools and features
Next

3. Enter **QueueSender** for the project name, **ServiceBusQueueQuickStart** for the solution name, and then select **Next**.


Configure your new project

Console Application C# Linux macOS Windows Console

Project name: QueueSender

Location: C:\Service Bus\

Solution name: ServiceBusQueueQuickStart

☐ Place solution and project in the same directory

4. On the **Additional information** page, select **Create** to create the solution and the project.

Add the NuGet packages to the project

Connection String

1. Select **Tools** > **NuGet Package Manager** > **Package Manager Console** from the menu.

2. Run the following command to install the **Azure.Messaging.ServiceBus** NuGet package:

PowerShell

```
Install-Package Azure.Messaging.ServiceBus
```

Add code to send messages to the queue

1. Replace the contents of `Program.cs` with the following code. The important steps are outlined in the following section, with additional information in the code

comments.

Connection string

- Creates a ServiceBusClient object using the connection string.
- Invokes the CreateSender method on the ServiceBusClient object to create a ServiceBusSender object for the specific Service Bus queue.
- Creates a ServiceBusMessageBatch object by using the ServiceBusSender.CreateMessageBatchAsync method.
- Add messages to the batch using the ServiceBusMessageBatch.TryAddMessage.
- Sends the batch of messages to the Service Bus queue using the ServiceBusSender.SendMessagesAsync method.

Important

Update placeholder values (`<NAMESPACE-CONNECTION-STRING>` and `<QUEUE-NAME>`) in the code snippet with names of your Service Bus namespace and queue.

C#

```
using Azure.Messaging.ServiceBus;

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

// the sender used to publish messages to the queue
ServiceBusSender sender;

// number of messages to be sent to the queue
const int numOfMessages = 3;

// The Service Bus client types are safe to cache and use as a sin-
gleton for the lifetime
// of the application, which is best practice when messages are be-
ing published or read
// regularly.
//
// set the transport type to AmqpWebSockets so that the
ServiceBusClient uses the port 443.
// If you use the default AmqpTcp, you will need to make sure that
the ports 5671 and 5672 are open

// TODO: Replace the <NAMESPACE-CONNECTION-STRING> and <QUEUE-NAME>
placeholders
```

```
var clientOptions = new ServiceBusClientOptions()
{
    TransportType = ServiceBusTransportType.AmqpWebSockets
};
client = new ServiceBusClient("<NAMESPACE-CONNECTION-STRING>",
clientOptions);
sender = client.CreateSender("<QUEUE-NAME>");

// create a batch
using ServiceBusMessageBatch messageBatch = await
sender.CreateMessageBatchAsync();

for (int i = 1; i <= numOfMessages; i++)
{
    // try adding a message to the batch
    if (!messageBatch.TryAddMessage(new ServiceBusMessage($"Message
{i}")))
    {
        // if it is too large for the batch
        throw new Exception($"The message {i} is too large to fit
in the batch.");
    }
}

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();
```

2. Build the project, and ensure that there are no errors.

3. Run the program and wait for the confirmation message.

Bash

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

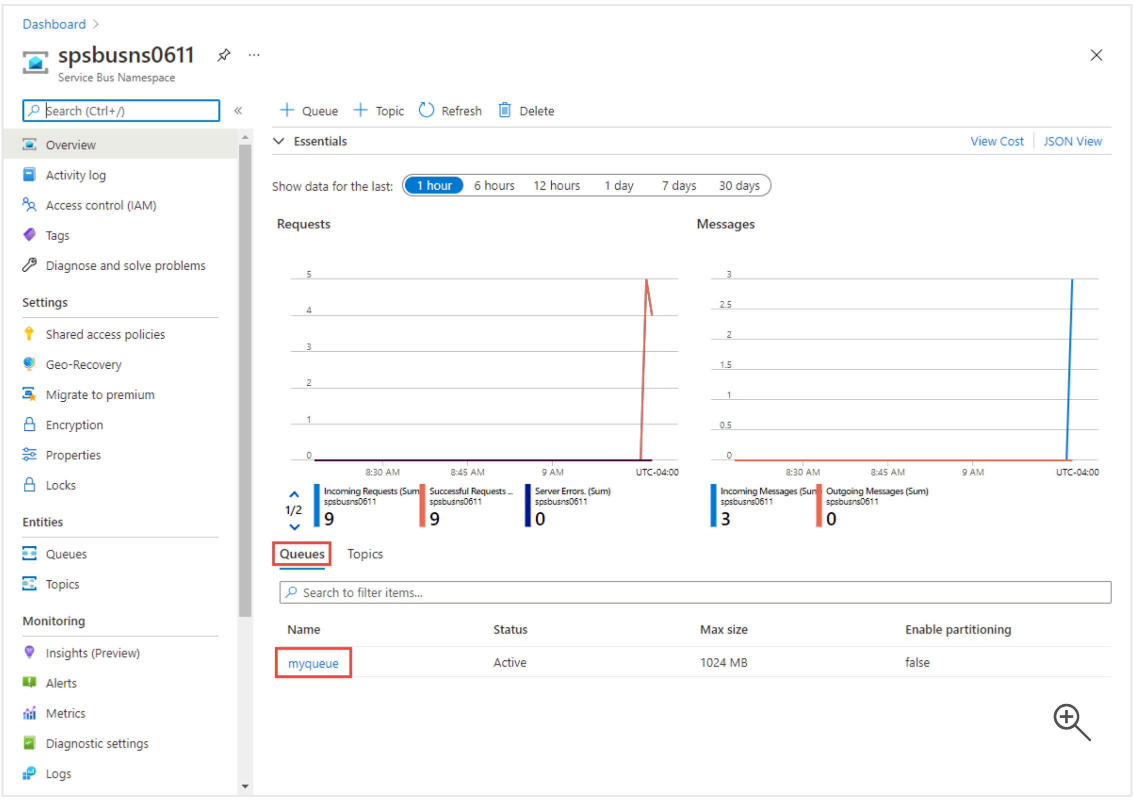
ⓘ **Important**

In most cases, it will take a minute or two for the role assignment to propagate in Azure. In rare cases, it might take up to **eight minutes**. If you receive authentication errors when you first run your code, wait a few moments and try again.

4. In the Azure portal, follow these steps:

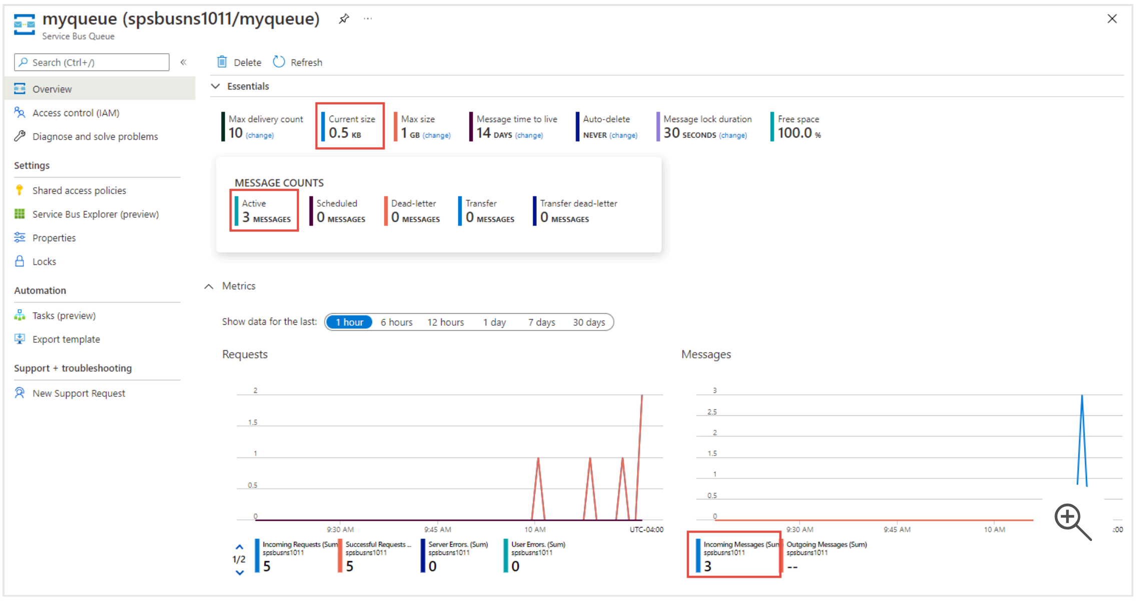
   a. Navigate to your Service Bus namespace.

   b. On the **Overview** page, select the queue in the bottom-middle pane.



Name	Status	Max size	Enable partitioning
myqueue	Active	1024 MB	false

   c. Notice the values in the **Essentials** section.



Max delivery count 10 (change) | Current size 0.5 KB | Max size 1 GB (change) | Message time to live 14 DAYS (change) | Auto-delete NEVER (change) | Message lock duration 30 SECONDS (change) | Free space 100.0 %

MESSAGE COUNTS
Active 3 MESSAGES | Scheduled 0 MESSAGES | Dead-letter 0 MESSAGES | Transfer 0 MESSAGES | Transfer dead-letter 0 MESSAGES

Incoming Requests (Sum) spsbusns1011 5 | Successful Requests... spsbusns1011 5 | Server Errors (Sum) spsbusns1011 0 | User Errors (Sum) spsbusns1011 0

Incoming Messages (Sum) spsbusns1011 3 | Outgoing Messages (Sum) spsbusns1011 --

Notice the following values:

- 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.

Receive messages from the queue

In this section, you create a .NET console application that receives messages from the queue.

> ⓘ **Note**
>
> This quickstart provides step-by-step instructions to implement a scenario of sending a batch of messages to a Service Bus queue and then receiving them. For more samples on other and advanced scenarios, see Service Bus .NET samples on GitHub .

Create a project for the receiver

1. In the Solution Explorer window, right-click the **ServiceBusQueueQuickStart** solution, point to **Add**, and select **New Project**.
2. Select **Console application**, and select **Next**.
3. Enter **QueueReceiver** for the **Project name**, and select **Create**.

4. In the **Solution Explorer** window, right-click **QueueReceiver**, and select **Set as a Startup Project**.

Add the NuGet packages to the project

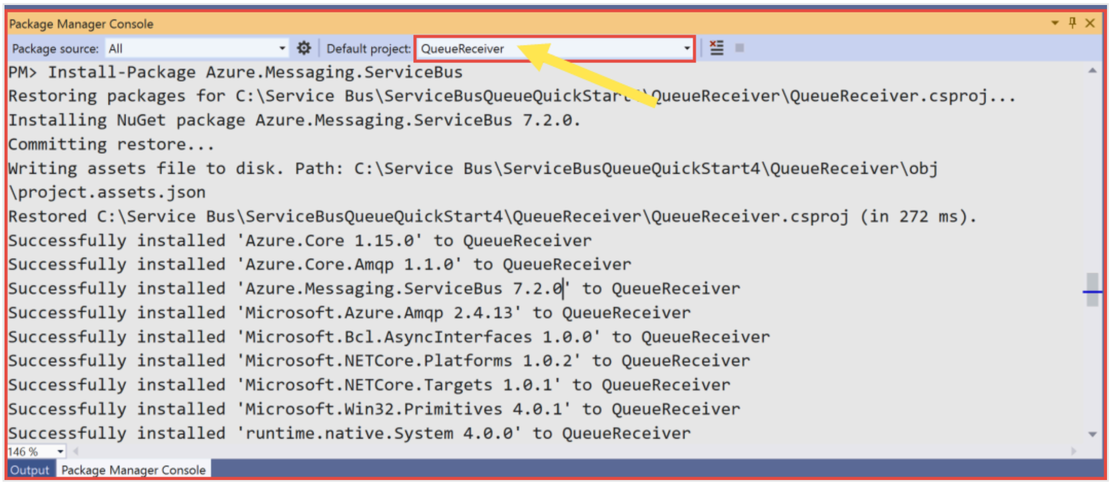
Connection String

1. Select **Tools** > **NuGet Package Manager** > **Package Manager Console** from the menu.

2. Run the following command to install the **Azure.Messaging.ServiceBus** NuGet package:

PowerShell

```powershell
Install-Package Azure.Messaging.ServiceBus
```



```
PM> Install-Package Azure.Messaging.ServiceBus
Restoring packages for C:\Service Bus\ServiceBusQueueQuickStart4\QueueReceiver\QueueReceiver.csproj...
Installing NuGet package Azure.Messaging.ServiceBus 7.2.0.
Committing restore...
Writing assets file to disk. Path: C:\Service Bus\ServiceBusQueueQuickStart4\QueueReceiver\obj
\project.assets.json
Restored C:\Service Bus\ServiceBusQueueQuickStart4\QueueReceiver\QueueReceiver.csproj (in 272 ms).
Successfully installed 'Azure.Core 1.15.0' to QueueReceiver
Successfully installed 'Azure.Core.Amqp 1.1.0' to QueueReceiver
Successfully installed 'Azure.Messaging.ServiceBus 7.2.0' to QueueReceiver
Successfully installed 'Microsoft.Azure.Amqp 2.4.13' to QueueReceiver
Successfully installed 'Microsoft.Bcl.AsyncInterfaces 1.0.0' to QueueReceiver
Successfully installed 'Microsoft.NETCore.Platforms 1.0.2' to QueueReceiver
Successfully installed 'Microsoft.NETCore.Targets 1.0.1' to QueueReceiver
Successfully installed 'Microsoft.Win32.Primitives 4.0.1' to QueueReceiver
Successfully installed 'runtime.native.System 4.0.0' to QueueReceiver
```

Add the code to receive messages from the queue

In this section, you add code to retrieve messages from the queue.

1. Within the `Program` class, add the following code:

Connection string

C#

```csharp
using System.Threading.Tasks;
using Azure.Messaging.ServiceBus;
```

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

// the processor that reads and processes messages from the queue
ServiceBusProcessor processor;
```

2. Append the following methods to the end of the `Program` class.

C#

```csharp
// handle received messages
async Task MessageHandler(ProcessMessageEventArgs args)
{
    string body = args.Message.Body.ToString();
    Console.WriteLine($"Received: {body}");

    // complete the message. message is deleted from the queue.
    await args.CompleteMessageAsync(args.Message);
}

// handle any errors when receiving messages
Task ErrorHandler(ProcessErrorEventArgs args)
{
    Console.WriteLine(args.Exception.ToString());
    return Task.CompletedTask;
}
```

3. Append the following code to the end of the `Program` class. The important steps are outlined in the following section, with additional information in the code comments.

Connection string

- Creates a ServiceBusClient object using the connection string.
- Invokes the CreateProcessor method on the ServiceBusClient object to create a ServiceBusProcessor object for the specified Service Bus queue.
- Specifies handlers for the ProcessMessageAsync and ProcessErrorAsync events of the ServiceBusProcessor object.
- Starts processing messages by invoking the StartProcessingAsync on the ServiceBusProcessor object.
- When user presses a key to end the processing, invokes the StopProcessingAsync on the ServiceBusProcessor object.

C#

```
// The Service Bus client types are safe to cache and use as a sin-
gleton for the lifetime
// of the application, which is best practice when messages are be-
ing published or read
// regularly.
//
// Set the transport type to AmqpWebSockets so that the
ServiceBusClient uses port 443.
// If you use the default AmqpTcp, make sure that ports 5671 and
5672 are open.

// TODO: Replace the <NAMESPACE-CONNECTION-STRING> and <QUEUE-NAME>
placeholders
var clientOptions = new ServiceBusClientOptions()
{
    TransportType = ServiceBusTransportType.AmqpWebSockets
};
client = new ServiceBusClient("<NAMESPACE-CONNECTION-STRING>",
clientOptions);

// create a processor that we can use to process the messages
// TODO: Replace the <QUEUE-NAME> placeholder
processor = client.CreateProcessor("<QUEUE-NAME>", 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();
}
```

4. The completed `Program` class should match the following code:

Connection string

C#

```
using Azure.Messaging.ServiceBus;
using System;
using System.Threading.Tasks;

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

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

// The Service Bus client types are safe to cache and use as a sin-
gleton for the lifetime
// of the application, which is best practice when messages are be-
ing published or read
// regularly.
//
// Set the transport type to AmqpWebSockets so that the
ServiceBusClient uses port 443.
// If you use the default AmqpTcp, make sure that ports 5671 and
5672 are open.

// TODO: Replace the <NAMESPACE-CONNECTION-STRING> and <QUEUE-NAME>
placeholders
var clientOptions = new ServiceBusClientOptions()
{
    TransportType = ServiceBusTransportType.AmqpWebSockets
};
client = new ServiceBusClient("<NAMESPACE-CONNECTION-STRING>",
clientOptions);

// create a processor that we can use to process the messages
// TODO: Replace the <QUEUE-NAME> placeholder
processor = client.CreateProcessor("<QUEUE-NAME>", 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();
}

// handle received messages
async Task MessageHandler(ProcessMessageEventArgs args)
{
    string body = args.Message.Body.ToString();
    Console.WriteLine($"Received: {body}");

    // complete the message. message is deleted from the queue.
    await args.CompleteMessageAsync(args.Message);
}

// handle any errors when receiving messages
Task ErrorHandler(ProcessErrorEventArgs args)
{
    Console.WriteLine(args.Exception.ToString());
    return Task.CompletedTask;
}
```

5. Build the project, and ensure that there are no errors.

6. Run the receiver application. You should see the received messages. Press any key to stop the receiver and the application.

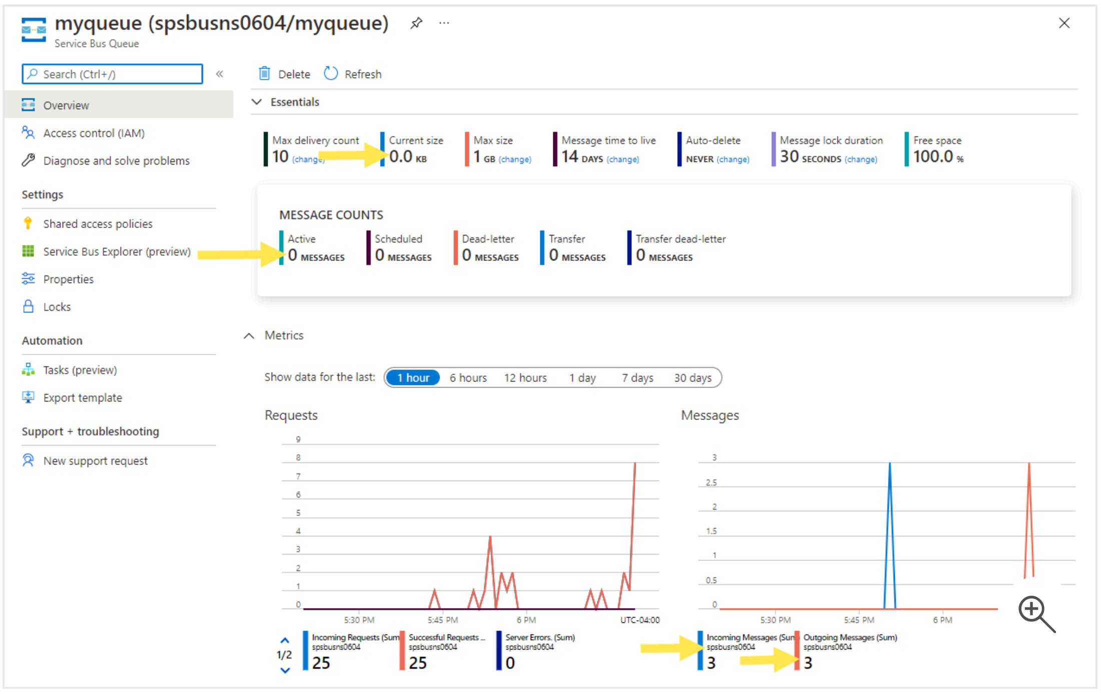
Console

```
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
```

7. Check the portal again. Wait for a few minutes and refresh the page if you don't see `0` for **Active** messages.

   - The **Active** message count and **Current size** values are now **0**.
   - In the **Messages** chart in the bottom **Metrics** section, you can see that there are three incoming messages and three outgoing messages for the queue.


myqueue (spsbusns0604/myqueue)
Service Bus Queue

Essentials

Max delivery count 10 (change)
Current size 0.0 KB
Max size 1 GB (change)
Message time to live 14 DAYS (change)
Auto-delete NEVER (change)
Message lock duration 30 SECONDS (change)
Free space 100.0 %

MESSAGE COUNTS
Active 0 MESSAGES
Scheduled 0 MESSAGES
Dead-letter 0 MESSAGES
Transfer 0 MESSAGES
Transfer dead-letter 0 MESSAGES

Metrics
Show data for the last: 1 hour

Incoming Requests (Sum) spsbusns0604 25
Successful Requests spsbusns0604 25
Server Errors. (Sum) spsbusns0604 0
Incoming Messages (Sum) spsbusns0604 3
Outgoing Messages (Sum) spsbusns0604 3

Clean up resources

Navigate to your Service Bus namespace in the Azure portal, and select **Delete** on the Azure portal to delete the namespace and the queue in it.

See also

See the following documentation and samples:

- Azure Service Bus client library for .NET - Readme
- Samples on GitHub
- .NET API reference
- Abstract away infrastructure concerns with higher-level frameworks like NServiceBus