

Service Bus Messaging Documentation

Azure Service Bus is a managed messaging service that helps different applications and services communicate with each other. It offers the following benefits:

- Distributes work evenly across multiple workers.
- Ensures safe data transfer between services and applications.
- Supports reliable, coordinated transactions that require high reliability.

Queues

Messages are sent to and received from **queues**. Queues store messages until the receiving application is available to receive and process them.

Prerequisites:

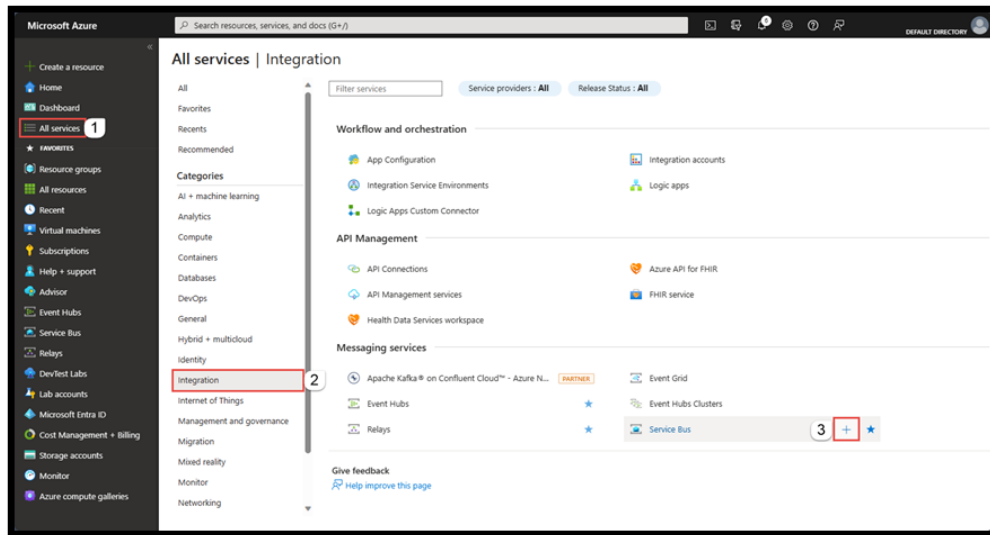
- **Azure subscription:** To use Azure services like Azure Service Bus, we need an Azure subscription. If we don't have one, we can sign up for a free trial.
- **Visual Studio 2022:** The sample application uses C# 10 features. We can still use older C# versions, but the syntax may differ. We recommend using .NET 6.0 or higher and Visual Studio 2022 or later for the latest features.

Create a namespace in the Azure portal

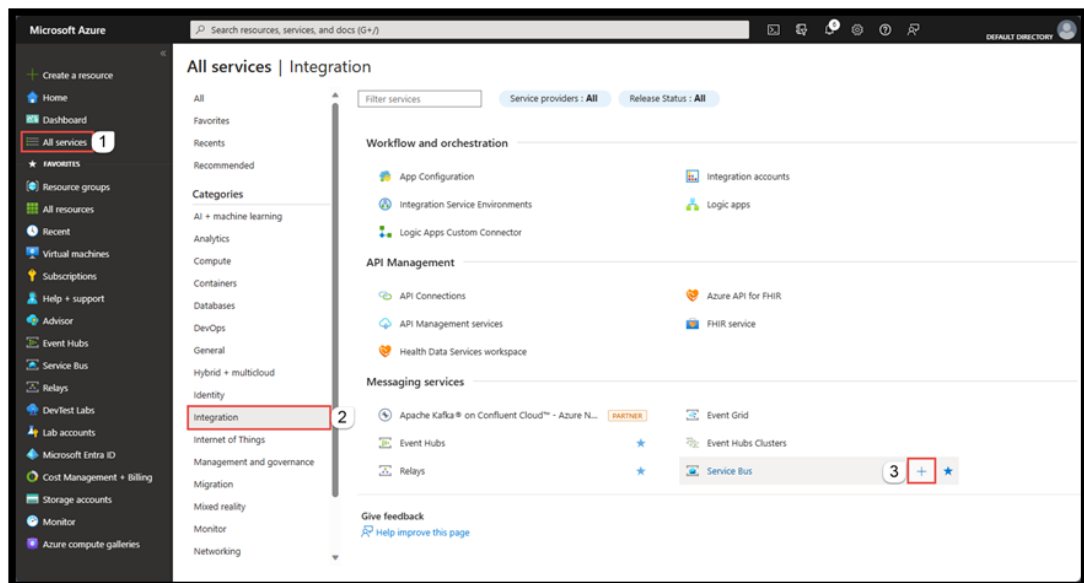
To start using Service Bus messaging in Azure, we first need to create a namespace with a unique name. A namespace acts as a container for Service Bus resources (like queues and topics) within our application.

To create a namespace:

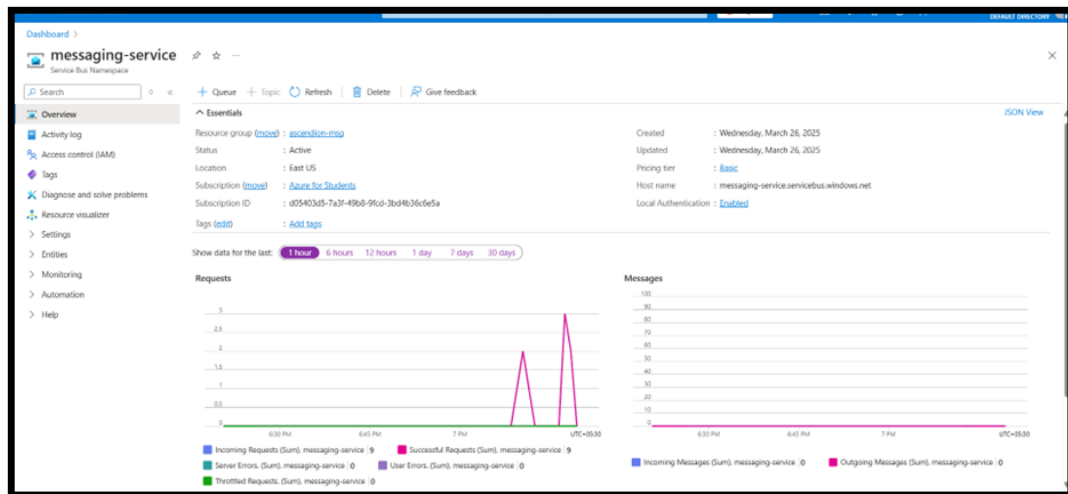
1. Sign in to the [Azure portal](#).
2. Navigate to the [All services](#) page.
3. In the left menu, select Integration, hover over Service Bus, and click the + button on the Service Bus tile.



4. Choose an Azure subscription to create the namespace in.
5. Select an existing resource group or create a new one.
6. Enter a unique name for the namespace (6 to 50 characters, only letters, numbers, and hyphens, and must start and end with a letter or number).
7. Pick a region for namespace.
8. Choose a pricing tier (select Basic).



9. Select **Review + create** at the bottom of the page.
10. On the **Review + create** page, review settings, and select **Create**.
11. Once the deployment of the resource is successful, select **Go to resource** on the deployment page.
12. Here, we see the home page for our Service Bus namespace.



Create a queue in the Azure portal

1. On the Service Bus Namespace page, expand "Entities" in the left menu and select "Queues."
2. On the Queues page, click "+ Queue" on the toolbar.
3. Enter a name for the queue and leave the default settings.
4. Click "Create."

Authenticate the app to Azure

Here, we have two ways to connect to Azure Service Bus: **passwordless** & using a **connection string**.

1. The first option uses Microsoft Entra ID and role-based access control (RBAC) to connect without needing a hard-coded connection string.
2. The second option uses a connection string, which might be easier for beginners.

I preferred the connection string option due to authentication issues.

Get the connection string

When you create a new namespace, an initial Shared Access Signature (SAS) policy is generated with primary and secondary keys and connection strings, giving full control over the namespace.

To copy the primary connection string:

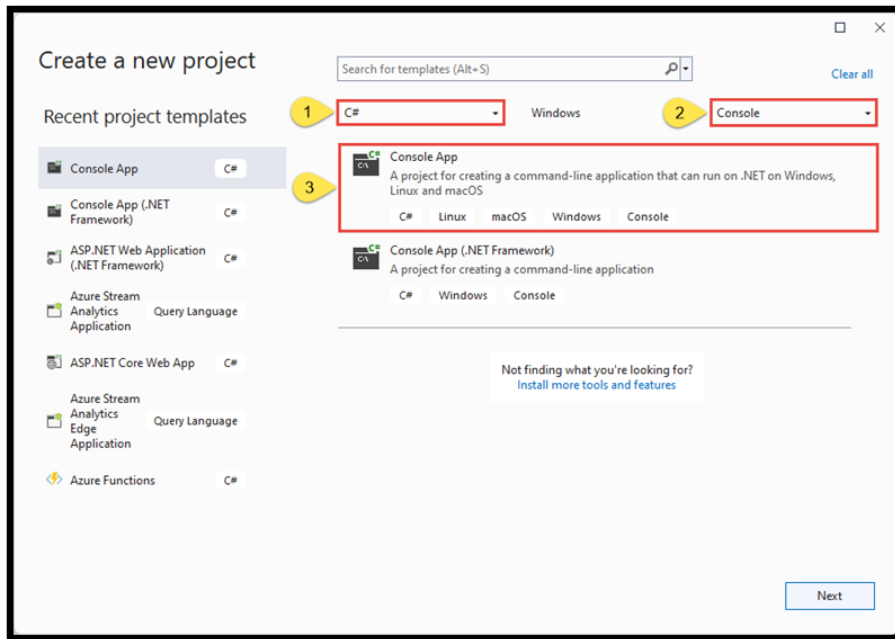
1. Go to the Service Bus Namespace page and select **Shared access policies**.
2. Click **RootManageSharedAccessKey**.
3. In the **Policy: RootManageSharedAccessKey** window, click the copy button next to **Primary Connection String** and paste it somewhere safe.

Send messages to the queue

Create a console application

- ❖ In Visual Studio, go to **File -> New -> Project**. If the dialog doesn't appear, select **File -> New -> Project** again.
- ❖ Select **C#** for the programming language.
- ❖ Select **Console** for the type of application.
- ❖ Select **Console App** from the results list.

- ❖ Then, select **Next**.



- ❖ Enter **QueueSender** for the project name, **ServiceBusQueueQuickStart** for the solution name, and then select **Next**.
- ❖ On the **Additional information** page, select **Create** to create the solution and the project.

Add the NuGet packages to the project

- Go to **Tools > NuGet Package Manager > Package Manager Console**.
- Run the command to install the **Azure.Messaging.ServiceBus** NuGet package.

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 key steps are:
 - Create a **ServiceBusClient** object using the connection string.

- Use the **CreateSender** method to create a **ServiceBusSender** for the queue.
- Create a **ServiceBusMessageBatch** using **CreateMessageBatchAsync**.
- Add messages to the batch with **TryAddMessage**.
- Send the batch of messages using **SendMessagesAsync**.

```
using Azure.Messaging.ServiceBus;
using QueueSender.Models;
using System;
using System.Collections.Generic;
using System.Text.Json;
using System.Threading.Tasks;

class Program
{
    static async Task Main(string[] args)
    {
        ServiceBusClient client;
        ServiceBusSender sender;

        const int numMessages = 1;

        var clientOptions = new ServiceBusClientOptions()
        {
            TransportType = ServiceBusTransportType.AmqpWebSockets
        };

        string connectionString = "Endpoint=sb://messaging-
service.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;Sh
aredAccessKey=pl8WJOC2oSTD5ApgCiLOqtn9i8adR0NCQ+ASbONuZ90=";

        client = new ServiceBusClient(connectionString, clientOptions);
        sender = client.CreateSender("myqueue");

        List<Contact> contactsToSend = new List<Contact>();

        try
        {
            using ServiceBusMessageBatch messageBatch = await
sender.CreateMessageBatchAsync();

            for (int i = 1; i <= numMessages; i++)
            {
```

```

        var contact = new Contact
        {
            FirstName = "John",
            LastName = "Doe",
            PhoneNumber = "+1234567890"
        };

        contactsToSend.Add(contact);

        string messageBody = JsonSerializer.Serialize(contact);

        if (!messageBatch.TryAddMessage(new ServiceBusMessage(messageBody)))
        {
            throw new Exception($"The message {i} is too large to fit in the batch.");
        }
    }

    await sender.SendMessagesAsync(messageBatch);

    Console.WriteLine($"Successfully sent a batch of {numOfMessages} message(s) to
the queue '{sender.EntityPath}' containing contacts:");

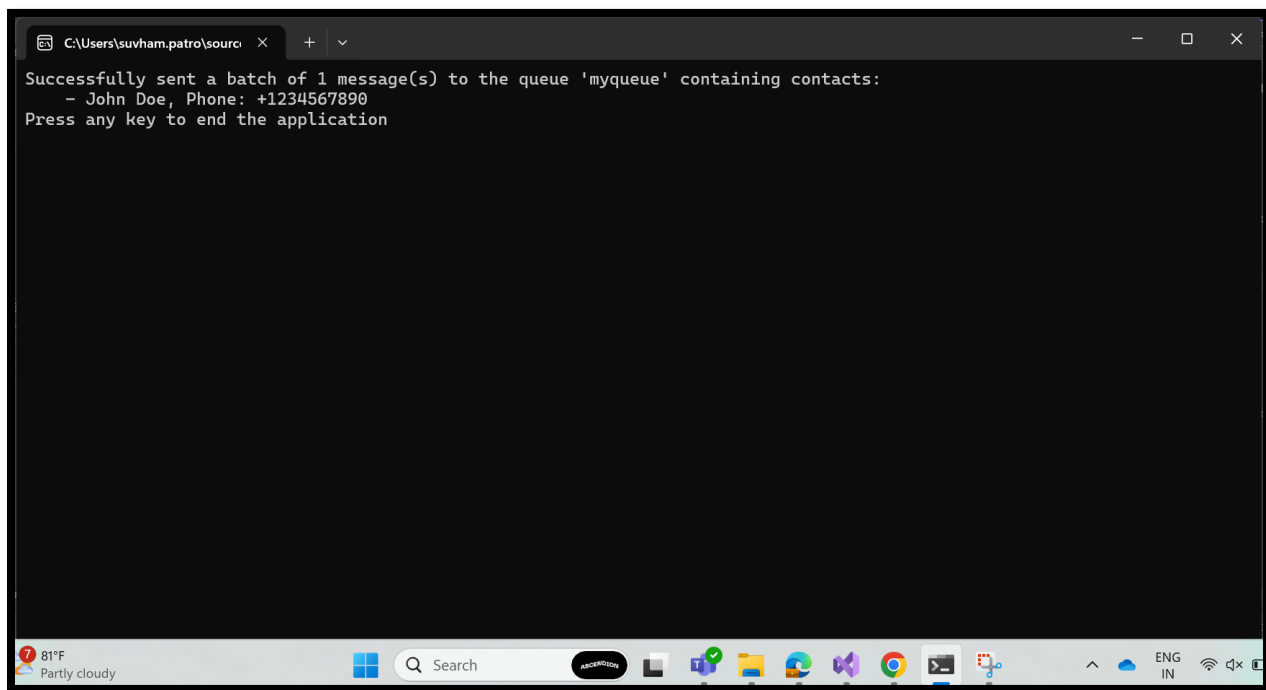
    foreach (var contact in contactsToSend)
    {
        Console.WriteLine($"  - {contact.FirstName} {contact.LastName}, Phone:
{contact.PhoneNumber}");
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
finally
{
    await sender.DisposeAsync();
    await client.DisposeAsync();
}

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

```

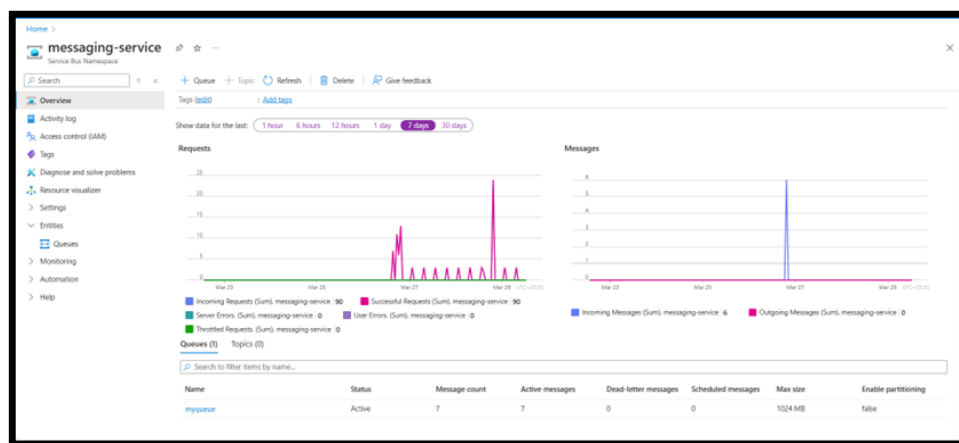
2. Build the project to check for errors.

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

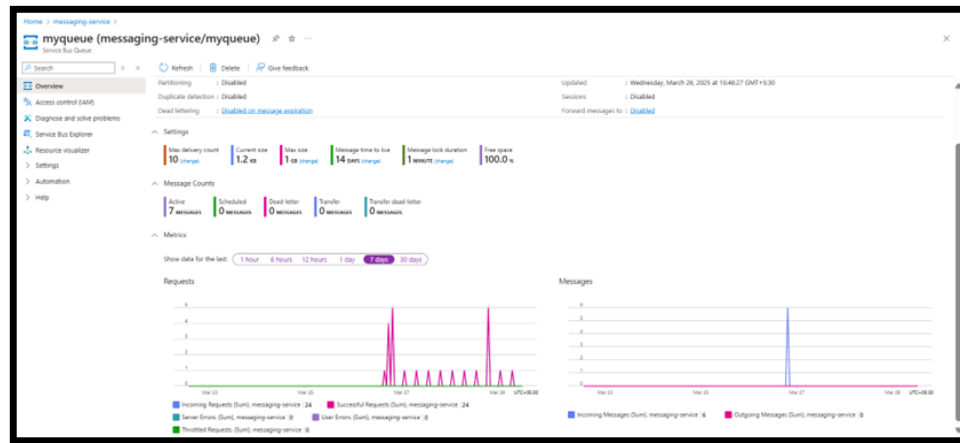


4. In the Azure portal, follow these steps:

- Go to your Service Bus namespace.
- On the Overview page, select the queue in the bottom-middle pane.



- Check the values in the Settings section.



Receive messages from the queue

Create a project for the receiver

- ❖ In the Solution Explorer, right-click on the **ServiceBusQueueQuickStart** solution.
- ❖ Hover over **Add**, then click **New Project**.
- ❖ Choose **Console Application** and click **Next**.
- ❖ Name the project **QueueReceiver**, then click **Create**.
- ❖ Right-click on **QueueReceiver** in the Solution Explorer and select **Set as Startup Project**.

Add the NuGet packages to the project

- Go to **Tools > NuGet Package Manager > Package Manager Console**.
- Run the command to install the **Azure.Messaging.ServiceBus** NuGet package.

Install-Package Azure.Messaging.ServiceBus

Add the code to receive messages from the queue

- **Create a ServiceBusClient** using the connection string.
- **Create a ServiceBusProcessor** by calling the CreateProcessor method on the ServiceBusClient, specifying the queue.
- Set up **event handlers** for ProcessMessageAsync (to handle messages) and ProcessErrorAsync (to handle errors).
- **Start processing** messages by calling StartProcessingAsync on the ServiceBusProcessor.
- When the user presses a key, **stop processing** by calling StopProcessingAsync on the ServiceBusProcessor.

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

class Program
{
    static async Task Main(string[] args)
    {
        string connectionString = "Endpoint=sb://messaging-
service.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;Sh
aredAccessKey=pl8WJOC2oSTD5ApgCiLOqtn9i8adR0NCQ+ASbONuZ90=";
        string queueName = "myqueue";

        var client = new ServiceBusClient(connectionString);
        var receiver = client.CreateReceiver(queueName);

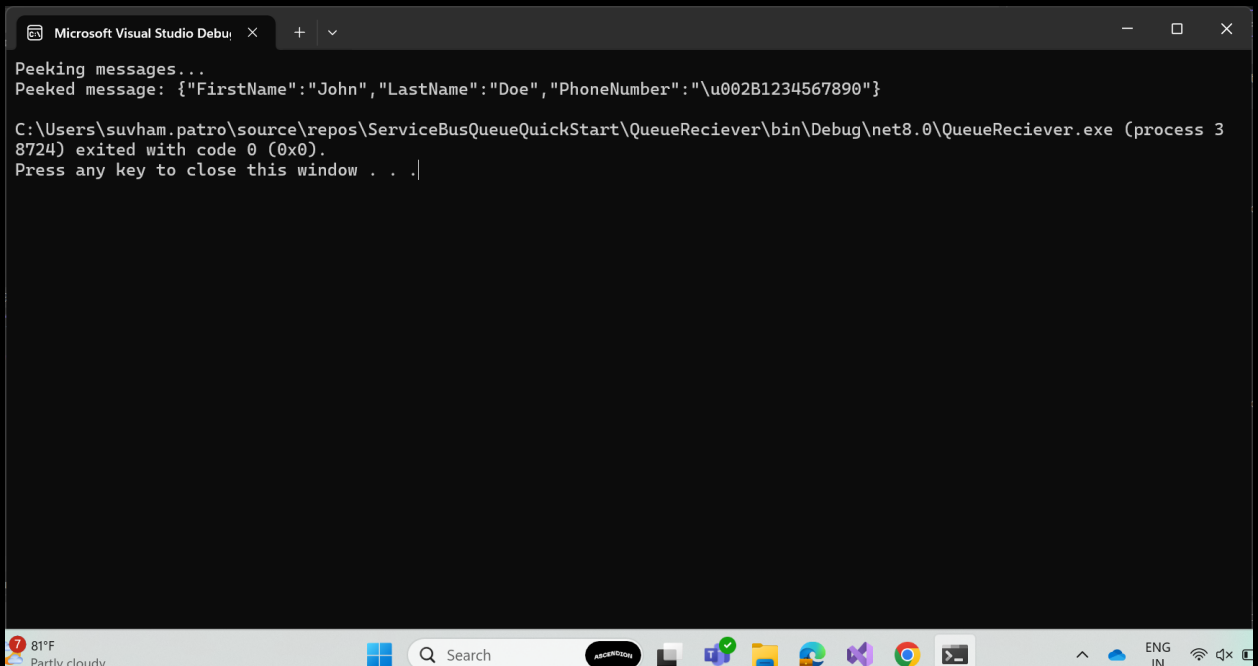
        try
        {
            Console.WriteLine("Peeking messages...");

            var messages = await receiver.PeekMessagesAsync(maxMessages: 10);

            if (messages.Count > 0)
            {
                foreach (var message in messages)
                {
                    Console.WriteLine($"Peeked message: {message.Body}");
                }
            }
            else
            {
                Console.WriteLine("No messages available in the queue.");
            }
        }
        catch (Exception ex)
        {
        }
```

```
        Console.WriteLine($"Error: {ex.Message}");
    }
    finally
    {
        await receiver.DisposeAsync();
        await client.DisposeAsync();
    }
}
```

- **Build the project** to make sure there are no errors.
- **Run the receiver application.** You should see the messages that are received.
- **Press any key** to stop the receiver and close the application.



```
Microsoft Visual Studio Debug Console
Peeking messages...
Peeked message: {"FirstName":"John","LastName":"Doe","PhoneNumber":"\u002B1234567890"}
C:\Users\suvham.patro\source\repos\ServiceBusQueueQuickStart\QueueReciever\bin\Debug\net8.0\QueueReciever.exe (process 38724) exited with code 0 (0x0).
Press any key to close this window . . .
```