This tutorial covers the following steps:

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

1. Create a namespace using the Azure portal

Note

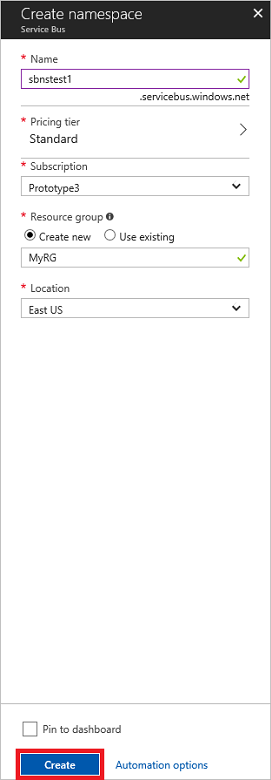
You can also create a Service Bus namespace and messaging entities using [PowerShell](https://docs.microsoft.com/en-us/powershell/azure/get-started-azureps). For more information, see [Use PowerShell to manage Service Bus resources](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-manage-with-ps).

If you have already created a Service Bus Messaging namespace, jump to the [Create a topic using the Azure portal](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-how-to-use-topics-subscriptions#2-create-a-topic-using-the-azure-portal) section.

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 addressing Service Bus resources within your application.

To create a namespace:

1. Log on to the [Azure portal](https://portal.azure.com/).
2. In the left navigation pane of the portal, click **+ Create a resource**, then click **Enterprise Integration**, and then click **Service Bus**.
3. In the **Create namespace** dialog, enter a namespace name. The system immediately checks to see if the name is available.
4. After making sure the namespace name is available, choose the pricing tier (Basic, Standard, or Premium).
5. In the **Subscription** field, choose an Azure subscription in which to create the namespace.
6. In the **Resource group** field, choose an existing resource group in which the namespace will live, or create a new one.
7. In **Location**, choose the country or region in which your namespace should be hosted.

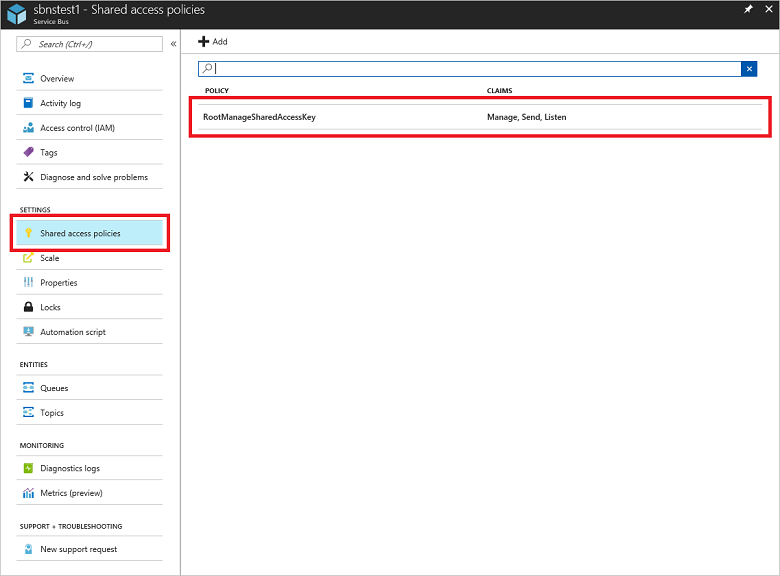


1. Click **Create**. The system now creates your namespace and enables it. You might have to wait several minutes as the system provisions resources for your account.

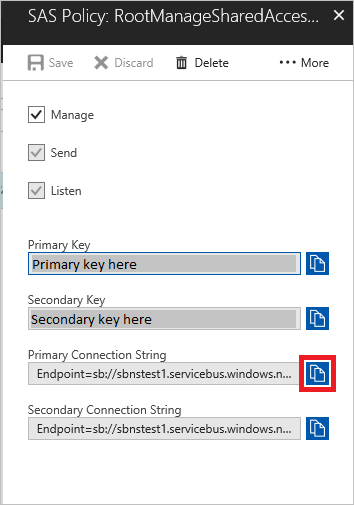
Obtain the management credentials

Creating a new namespace automatically generates an initial Shared Access Signature (SAS) rule with an associated pair of primary and secondary keys that each grant full control over all aspects of the namespace. See [Service Bus authentication and authorization](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-authentication-and-authorization) for information about how to create further rules with more constrained rights for regular senders and receivers. To copy the initial rule, follow these steps:

1. Click **All resources**, then click the newly created namespace name.
2. In the namespace window, click **Shared access policies**.
3. In the **Shared access policies** screen, click **RootManageSharedAccessKey**.



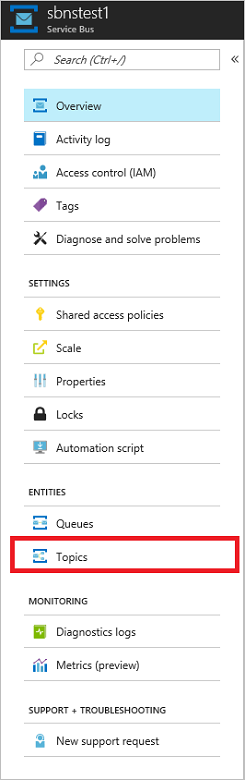
1. In the **Policy: RootManageSharedAccessKey** window, click the copy button next to **Connection string–primary key**, to copy the connection string to your clipboard for later use. Paste this value into Notepad or some other temporary location.



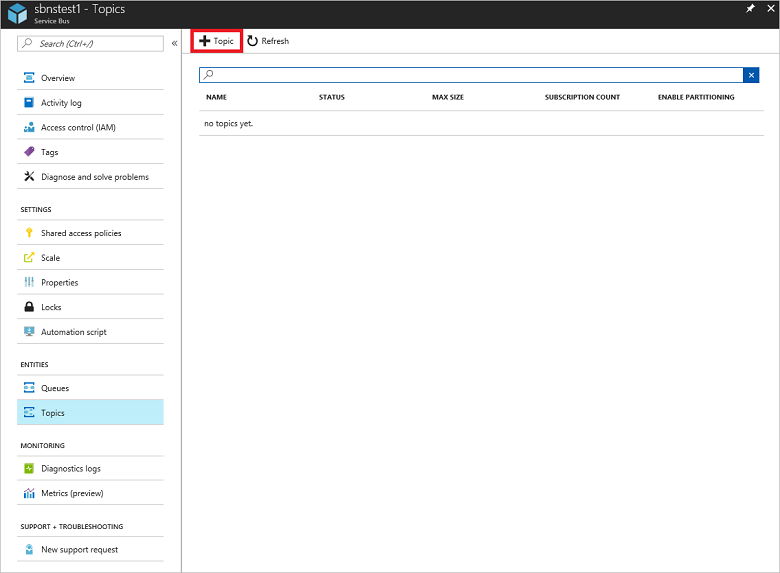
1. Repeat the previous step, copying and pasting the value of **Primary key** to a temporary location for later use.

2. Create a topic using the Azure portal

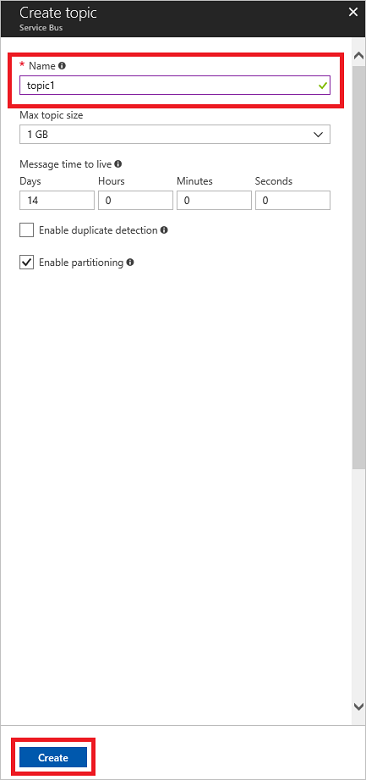
1. Log on to the [Azure portal](https://portal.azure.com/).
2. In the left navigation pane of the portal, click **Service Bus** (if you don't see **Service Bus**, click **All services**, or click on **All resources**). Click the namespace in which you would like to create the topic.
3. The namespace overview window opens. Click **Topics**:



1. Click **+ Topic**.



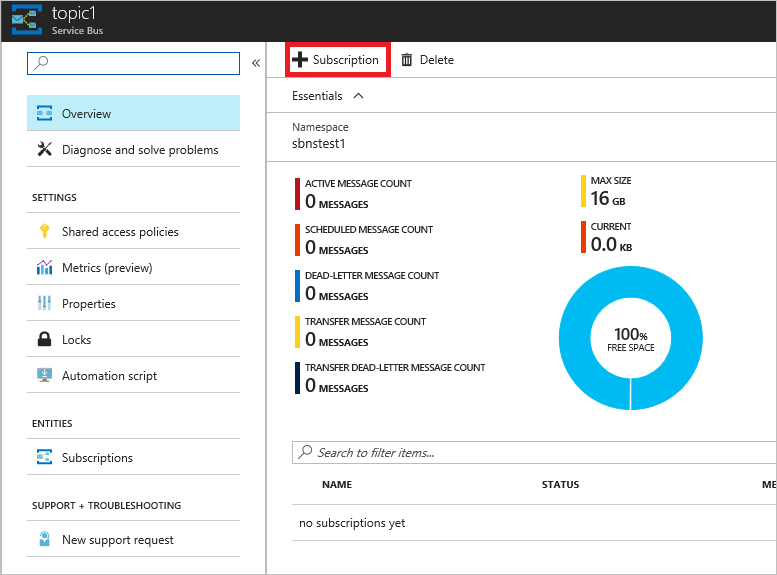
1. Enter a name for the topic. Leave the other options with their default values.



1. At the bottom of the dialog, click **Create**.

3. Create a subscription to the topic

1. In the portal resources pane, click the namespace you created in step 1, then click **Topics**, and then click name of the topic you created in step 2.
2. At the top of the overview pane, click **+ Subscription** to add a subscription to this topic.



1. Enter a name for the subscription. Leave the other options with their default values.

4. Send messages to the topic

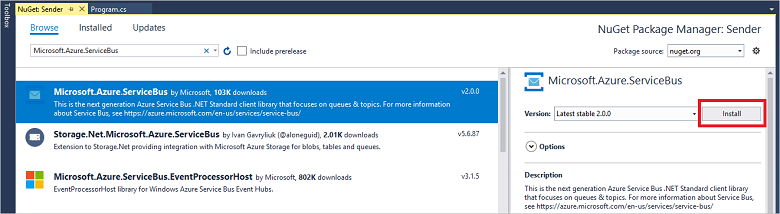
To send messages to the topic, write a C# console application using Visual Studio.

Create a console application

Launch Visual Studio and create a new **Console App (.NET Core)** project.

Add the Service Bus NuGet package

1. Right-click the newly created project and select **Manage NuGet Packages**.
2. Click the **Browse** tab, search for **[Microsoft.Azure.ServiceBus](https://www.nuget.org/packages/Microsoft.Azure.ServiceBus/)**, and then select the **Microsoft.Azure.ServiceBus** item. Click **Install** to complete the installation, then close this dialog box.



1. Here is what your sender Program.cs file should look like.

C#Copy

namespace CoreSenderApp

{

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 TopicName = "<your\_topic\_name>";

static ITopicClient topicClient;

static void Main(string[] args)

{

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

}

static async Task MainAsync()

{

const int numberOfMessages = 10;

topicClient = new TopicClient(ServiceBusConnectionString, TopicName);

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

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

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

// Send messages.

await SendMessagesAsync(numberOfMessages);

Console.ReadKey();

await topicClient.CloseAsync();

}

static async Task SendMessagesAsync(int numberOfMessagesToSend)

{

try

{

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

{

// Create a new message to send to the topic

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 topic

await topicClient.SendAsync(message);

}

}

catch (Exception exception)

{

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

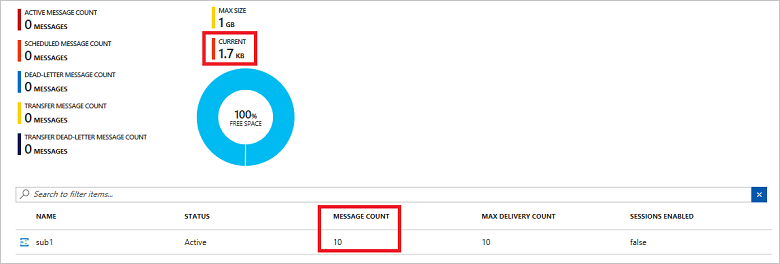
}

}

}

}

1. Run the program, and check the Azure portal: click the name of your topic in the namespace **Overview** window. The topic **Essentials** screen is displayed. In the subscription listed near the bottom of the window, notice that the **Message Count** value for the subscription is now **10**. Each time you run the sender application without retrieving the messages (as described in the next section), this value increases by 10. Also note that the current size of the topic increments the **Current** value in the **Essentials** window each time the app adds messages to the topic.



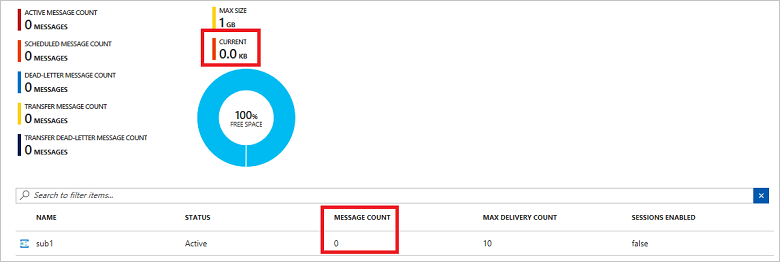
## Receive messages from the subscription

To receive the messages you just sent, create another .NET Core console application and install the **Microsoft.Azure.ServiceBus** NuGet package, similar to the previous sender application.

### Write code to receive messages from the subscription

1. namespace CoreReceiverApp
2. {
3. using System;
4. using System.Text;
5. using System.Threading;
6. using System.Threading.Tasks;
7. using Microsoft.Azure.ServiceBus;
8. class Program
9. {
10. const string ServiceBusConnectionString = "<your\_connection\_string>";
11. const string TopicName = "<your\_topic\_name>";
12. const string SubscriptionName = "<your\_subscription\_name>";
13. static ISubscriptionClient subscriptionClient;
14. static void Main(string[] args)
15. {
16. MainAsync().GetAwaiter().GetResult();
17. }
18. static async Task MainAsync()
19. {
20. subscriptionClient = new SubscriptionClient(ServiceBusConnectionString, TopicName, SubscriptionName);
21. Console.WriteLine("======================================================");
22. Console.WriteLine("Press ENTER key to exit after receiving all the messages.");
23. Console.WriteLine("======================================================");
24. // Register subscription message handler and receive messages in a loop.
25. RegisterOnMessageHandlerAndReceiveMessages();
26. Console.ReadKey();
27. await subscriptionClient.CloseAsync();
28. }
29. static void RegisterOnMessageHandlerAndReceiveMessages()
30. {
31. // Configure the message handler options in terms of exception handling, number of concurrent messages to deliver, etc.
32. var messageHandlerOptions = new MessageHandlerOptions(ExceptionReceivedHandler)
33. {
34. // Maximum number of concurrent calls to the callback ProcessMessagesAsync(), set to 1 for simplicity.
35. // Set it according to how many messages the application wants to process in parallel.
36. MaxConcurrentCalls = 1,
37. // Indicates whether MessagePump should automatically complete the messages after returning from User Callback.
38. // False below indicates the Complete will be handled by the User Callback as in `ProcessMessagesAsync` below.
39. AutoComplete = false
40. };
41. // Register the function that processes messages.
42. subscriptionClient.RegisterMessageHandler(ProcessMessagesAsync, messageHandlerOptions);
43. }
44. static async Task ProcessMessagesAsync(Message message, CancellationToken token)
45. {
46. // Process the message.
47. Console.WriteLine($"Received message: SequenceNumber:{message.SystemProperties.SequenceNumber} Body:{Encoding.UTF8.GetString(message.Body)}");
48. // Complete the message so that it is not received again.
49. // This can be done only if the subscriptionClient is created in ReceiveMode.PeekLock mode (which is the default).
50. await subscriptionClient.CompleteAsync(message.SystemProperties.LockToken);
51. // Note: Use the cancellationToken passed as necessary to determine if the subscriptionClient has already been closed.
52. // If subscriptionClient has already been closed, you can choose to not call CompleteAsync() or AbandonAsync() etc.
53. // to avoid unnecessary exceptions.
54. }
55. static Task ExceptionReceivedHandler(ExceptionReceivedEventArgs exceptionReceivedEventArgs)
56. {
57. Console.WriteLine($"Message handler encountered an exception {exceptionReceivedEventArgs.Exception}.");
58. var context = exceptionReceivedEventArgs.ExceptionReceivedContext;
59. Console.WriteLine("Exception context for troubleshooting:");
60. Console.WriteLine($"- Endpoint: {context.Endpoint}");
61. Console.WriteLine($"- Entity Path: {context.EntityPath}");
62. Console.WriteLine($"- Executing Action: {context.Action}");
63. return Task.CompletedTask;
64. }
65. }
66. }

Run the program, and check the portal again. Notice that the **Message Count** and **Current** values are now **0**.



Congratulations! Using the .NET Standard library, you have now created a topic and subscription, sent 10 messages, and received those messages.