

# Exercise - Write code that uses Service Bus topics

10 minutes

Sandbox activated! Time remaining: 2 hr 50 min

You have used 4 of 10 sandboxes for today. More sandboxes will be available tomorrow.

You have chosen to use an Azure Service Bus topic to distribute messages about sales performance in your sales force distributed application. The app used by sales personnel on their mobile devices will send messages that summarize sales figures for each area and time period. Those messages will be distributed to web services located in the company's operational regions, including the Americas and Europe.

You have already implemented the necessary infrastructure in your Azure subscription, including the topic and subscriptions. Now, you want to write the code that sends messages to the topic and retrieves messages from a subscription. Before you begin, you'll need to make sure you are working in the correct directory by executing the following command in the Cloud Shell:

Bash

 Copy

```
cd ~/mslearn-connect-services-together/implement-message-workflows-with-service-bus/src/start
```

## Configure a connection string to a Service Bus namespace

Start by configuring connection strings both in the sending and receiving components. You'll use the same connection string as before. If you don't still have it, refer back to the previous exercise to learn how to retrieve it.

1. In the editor, open **performancemessagesender/Program.cs** and locate the following line of code:

C#

 Copy

```
const string ServiceBusConnectionString = "";
```


Paste the connection string between the quotation marks and save the file either through the "..." menu, or the accelerator key (**Ctrl+S** on Windows and Linux, **Cmd+S** on macOS).

2. Repeat the previous step in **performancemessagereceiver/Program.cs**, pasting in the same connection string value and save the file.


## Write code that sends a message to the topic

To complete the component that sends messages about sales performance, follow these steps:


1. Open **performancemessagesender/Program.cs** in the editor.
2. Locate the `SendPerformanceMessageAsync()` method.
3. Within that method, locate the following line of code:

C#	 Copy
<pre>// Create a Topic Client here</pre>	


4. To create a topic client, replace that line of code with the following code:

C#	 Copy
<pre>topicClient = new TopicClient(ServiceBusConnectionString, TopicName);</pre>	


5. Within the `try...catch` block, locate the following line of code:

C#	 Copy
<pre>// Create and send a message here</pre>	


6. To create and format a message for the queue, replace that line of code with the following code:

C#	 Copy
<pre>string messageBody = \$"Total sales for Brazil in August: \$13m."; var message = new Message(Encoding.UTF8.GetBytes(messageBody));</pre>	


7. To display the message in the console, on the next line, add the following code:

C#	 Copy
<pre>Console.WriteLine(\$"Sending message: {messageBody}");</pre>	


8. To send the message to the queue, on the next line, add the following code:

C#	 Copy
<pre>await topicClient.SendAsync(message);</pre>	

9. Locate the following line of code:

C#	 Copy
<pre>// Close the connection to the topic here</pre>	


10. To close the connection to Service Bus, replace that line of code with the following code:

C#	 Copy
<pre>await topicClient.CloseAsync();</pre>	

11. Save the file.


## Send a message to the topic

To run the component that sends a message about a sale, run the following command in the Cloud Shell:

Bash	 Copy
<pre>dotnet run -p performancemessagesender</pre>	

As the program executes, you'll see messages printed indicating that it's sending a message. Each time you run the app, one additional message will be added to the topic and each subscriber will receive a copy.

Once it's finished, run the following command to see how many messages are in the Americas subscription:

Azure CLI	 Copy
<pre>az servicebus topic subscription show \   --resource-group learn-d6156237-49fa-486e-b28a-133de5188629 \</pre>	


```
--namespace-name <namespace-name> \  
--topic-name salesperformancemessages \  
--name Americas \  
--query messageCount
```

If you substitute `EuropeAndAfrica` for `Americas`, you should see that both subscriptions have the same number of messages.


## Write code that receives a message from a topic subscription

To complete the component that retrieves messages about sales performance, follow these steps:


1. Open **performancemessagereceiver/Program.cs** in the editor.
2. Locate the `MainAsync()` method.
3. Within that method, locate the following line of code:

C#	 Copy
<pre>// Create a subscription client here</pre>	


4. To create a subscription client, replace that line with the following code:

C#	 Copy
<pre>subscriptionClient = new SubscriptionClient(ServiceBusConnectionString, TopicName, SubscriptionName);</pre>	

5. Locate the `RegisterMessageHandler()` method.
6. To configure message handling options, replace all the code within that method with the following code:


C#	 Copy
<pre>var messageHandlerOptions = new MessageHandlerOptions(ExceptionReceivedHandler) {     MaxConcurrentCalls = 1,     AutoComplete = false };</pre>	

7. To register the message handler, on the next line, add the following code:


C#	 Copy
<pre>subscriptionClient.RegisterMessageHandler(ProcessMessagesAsync, messageHandlerOptions);</pre>	

8. Locate the `ProcessMessagesAsync()` method. You have registered this method as the one that handles incoming messages.


9. To display incoming messages in the console, replace all the code within that method with the following code:

C#	 Copy
<pre>Console.WriteLine(\$"Received sale performance message: SequenceNumber: {message.SystemProperties.SequenceNumber} Body: {Encoding.UTF8.GetString(message.Body)}");</pre>	


10. To remove the received message from the subscription, on the next line, add the following code:

C#	 Copy
<pre>await subscriptionClient.CompleteAsync(message.SystemProperties.LockToken);</pre>	

11. Return to the `MainAsync()` method and locate the following line of code:

C#	 Copy
<pre>// Close the subscription here</pre>	

12. To close the connection to Service Bus, replace that code with the following code:

C#	 Copy
<pre>await subscriptionClient.CloseAsync();</pre>	

13. In Visual Studio Code, close all editor windows and save all changed files.

## Retrieve a message from a topic subscription

To run the component that retrieves a message about sales performance, follow these steps:

Bash

 Copy

```
dotnet run -p performancemessagereceiver
```

When the program stops printing notifications that it is receiving messages, press `Enter` to stop the app. Then, run the same command as before to confirm that there are zero remaining messages in the `Americas` subscription.

Azure CLI

 Copy

```
az servicebus topic subscription show \  
  --resource-group learn-d6156237-49fa-486e-b28a-133de5188629 \  
  --namespace-name <namespace-name> \  
  --topic-name salesperformancemessages \  
  --name Americas \  
  --query messageCount
```

If you substitute `EuropeAndAfrica` for `Americas`, you'll see that the message count has not changed. The application only received messages from the `Americas` subscription.

## Next unit: Summary

[Continue >](#) English (United States)  Theme[Previous Version Docs](#)[Blog](#)[Contribute](#)[Privacy & Cookies](#)[Terms of Use](#)[Trademarks](#)

© Microsoft 2020



Azure Cloud Shell

```
=====
Press ENTER key to exit after receiving all
=====

rajani_net@Azure:~/mslearn-connect-services-
with-service-bus/src/start$ az servicebus top
> --resource-group learn-d6156237-49fa-4
> --namespace-name salesteamappr21may20
```