100 XP

Exercise - Write code that uses Service Bus queues

10 minutes

Sandbox activated! Time remaining: **25 min**

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

You've chosen to use a Service Bus queue to exchange messages about individual sales between the mobile app that your sales personnel use and the web service, hosted in Azure, that will store details about each sale in an Azure SQL Database instance.

You've already implemented the necessary objects in your Azure subscription. Now, you want to write code that sends messages to that queue and retrieves messages.

Clone and open the starter application

In this unit, you'll build two console applications. The first application places messages into a Service Bus queue and the second retrieves them. The applications are part of a single .NET Core solution.

1. Start by cloning the solution: run the following commands in the Cloud Shell:

```bash
cd ~
git clone https://github.com/MicrosoftDocs/mslearn-connect-services-together.git
```

2. Next, change directories into the starter folder and open the Cloud Shell editor.

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

Configure a connection string to a Service Bus namespace

In order to access a Service Bus namespace and use a queue, you must configure two pieces of information in your console apps:

- The endpoint for your namespace
- The shared access key for authentication

Both of these values can be obtained from the Azure portal in the form of a complete connection string.

> **Note**
>
> For simplicity, you will hard-code the connection string in the **Program.cs** file of both console applications. In a production application, you might use a configuration file or even Azure Key Vault to store the connection string.

1. Run the following command in the CloudShell to display the primary connection string for your Service Bus namespace. Replace `<namespace-name>` with the name of your Service Bus namespace.

   Azure CLI — Copy

   ```
   az servicebus namespace authorization-rule keys list \
       --resource-group learn-64b418db-f43b-4f90-ac43-7f877b571aaf \
       --name RootManageSharedAccessKey \
       --query primaryConnectionString \
       --output tsv \
       --namespace-name <namespace-name>
   ```

   You'll need this connection string multiple times throughout this module, so you might want to paste it somewhere handy.

2. Copy the key from Cloud Shell. In the editor, open **privatemessagesender/Program.cs** and locate the following line of code:

   C# — Copy

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

   Paste the connection string between the quotation marks. Save the file using the `Ctrl+S` keys.

3. Repeat the previous step in **privatemessagereceiver/Program.cs**, pasting in the same connection string value. Save the file either through the "..." menu, or the accelerator key (`Ctrl+S` on Windows and Linux, `Cmd+S` on macOS).

Write code that sends a message to the queue

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

1. Open **privatemessagesender/Program.cs** in the editor.

2. Locate the `SendSalesMessageAsync()` method.

3. Within that method, locate the following line of code:

   C# Copy

   ```csharp
   // Create a queue client here
   ```

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

   C# Copy

   ```csharp
   queueClient = new QueueClient(ServiceBusConnectionString, QueueName);
   ```

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

   C# Copy

   ```csharp
   // Create and send a message here
   ```

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

   C# Copy

   ```csharp
   string messageBody = $"$10,000 order for bicycle parts from retailer
   Adventure Works.";
   var message = new Message(Encoding.UTF8.GetBytes(messageBody));
   ```

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

   C# Copy

   ```csharp
   Console.WriteLine($"Sending message: {messageBody}");
   ```

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

   C# Copy

   ```csharp
   await queueClient.SendAsync(message);
   ```

9. Locate the following line of code:

   C# Copy

   ```csharp
   // Close the connection to the queue here
   ```

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

    C# Copy

    ```csharp
    await queueClient.CloseAsync();
    ```

11. Save the file.

Send a message to the queue

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

Bash Copy

```bash
dotnet run -p privatemessagesender
```

Note

The apps you run during this exercise may take a moment to start up, as `dotnet` has to restore packages from remote sources and build the apps the first time they are run.

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 queue.

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

Azure CLI Copy

```
az servicebus queue show \
    --resource-group learn-64b418db-f43b-4f90-ac43-7f877b571aaf \
    --name salesmessages \
    --query messageCount \
    --namespace-name <namespace-name>
```

Write code that receives a message from the queue

1. Open **privatemessagereceiver/Program.cs** in the editor

2. Locate the `ReceiveSalesMessageAsync()` method.

3. Within that method, locate the following line of code:

   C# Copy

   ```csharp
   // Create a queue client here
   ```

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

   C# Copy

   ```csharp
   queueClient = new QueueClient(ServiceBusConnectionString, QueueName);
   ```

5. Locate the `RegisterMessageHandler()` method.

6. To configure message handling options, replace all the code within that method with the following code:

   C# Copy

   ```csharp
   var messageHandlerOptions = new
   MessageHandlerOptions(ExceptionReceivedHandler)
   {
       MaxConcurrentCalls = 1,
       AutoComplete = false
   };
   ```

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

   C# Copy

   ```csharp
   queueClient.RegisterMessageHandler(ProcessMessagesAsync,
   messageHandlerOptions);
   ```

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

   ```csharp
   Console.WriteLine($"Received message: SequenceNumber:
   {message.SystemProperties.SequenceNumber} Body:
   {Encoding.UTF8.GetString(message.Body)}");
   ```

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

    C# Copy

    ```csharp
    await queueClient.CompleteAsync(message.SystemProperties.LockToken);
    ```

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

    C# Copy

    ```csharp
    // Close the queue here
    ```

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

    C# Copy

    ```csharp
    await queueClient.CloseAsync();
    ```

13. Save the file.

Retrieve a message from the queue

To run the component that receives a message about a sale, run this command in the Cloud Shell:

Bash Copy

```bash
dotnet run -p privatemessagereceiver
```

When you see that the message has been received and displayed in the console, press `Enter` to stop the app. Then, run the same command as before to confirm that all of the messages have been removed from the queue:

Azure CLI Copy

```
az servicebus queue show \
    --resource-group learn-64b418db-f43b-4f90-ac43-7f877b571aaf \
    --name salesmessages \
    --query messageCount \
    --namespace-name <namespace-name>
```

This will show `0` if all the messages have been removed.

You have written code that sends a message about individual sales to a Service Bus queue. In the sales force distributed application, you should write this code in the mobile app that sales personnel use on devices.

You have also written code that receives a message from the Service Bus queue. In the sales force distributed application, you should write this code in the web service that runs in Azure and processes received messages.

Next unit: Write code that uses Service Bus topics

Continue >

Azure Cloud Shell