# Module 8: Designing a Communication Strategy by Using Queues and Service Bus

# Lab: Using Queues and Service Bus to Manage Communication Between Web Applications in Azure

### Scenario

Now that you can generate sign-in sheets in worker roles, you need a scalable and consistent way to enqueue messages for the worker role. You have decided to use an Azure queue mechanism so that you can scale the worker roles in isolation to meet the demand of the queue size. In this lab, you will begin by implementing the communication between the Administration web application and the worker role by using Storage queues. Then you will replace that implementation with an implementation that uses Service Bus queues.

### Objectives

After you complete this lab, you will be able to:

* Create a Service Bus namespace by using the Portal.
* Create Storage queue messages.
* Consume Azure Storage queue messages.
* Create Service Bus queue messages.
* Consume Service Bus queue messages.

### Lab Setup

**Estimated Time: 60 minutes**

Before starting this lab, you must complete the lab in Module 2. For the lab in this module, you will use the available host machine. Also, you must complete the following steps:

1. On the host computer, click **Start**, type **Remote**, and then click **Remote Desktop Connection**.
2. In Remote Desktop Connection, provide the name of your virtual machine in the **Computer** box by using the following format:
   * **[Your VM IP Address]:[*Your VM RDP Port*]**

* **Note:** The name and port for your virtual machine might be saved in the Computer drop-down list. If this is the case, use this value instead of typing it in manually. If you are unsure about your virtual machine’s RDP port, use either of the Azure portals to find your virtual machine’s endpoints. The endpoint with the name **Remote Desktop** is the correct port for RDP. This port is randomized to protect your virtual machine from unauthorized access.

1. In Remote Desktop Connection, click **Connect**. Wait until the RDP client accesses the virtual machine.
2. If necessary, sign in by using the following credentials:
   * User name: **Student**
   * Password: **AzurePa$$w0rd**
3. Verify that you received the credentials to sign in to the Azure portal from your training provider. You will use these credentials and the Azure account throughout the labs in this course.

## Exercise 1: Creating an Azure Service Bus Namespace

### Scenario

In this exercise, you will:

* Create a Service Bus namespace by using the Portal.

The main tasks for this exercise are as follows:

1. Create the Service Bus namespace by using the Portal.

#### Task 1: Create the Service Bus namespace by using the Portal

**Note:** Service Bus functionality is not available yet in the Azure Portal. Because of this, the classic Portal will be used for this lab.

1. Sign in to the Classic Azure Portal ([*https://manage.windowsazure.com*](https://manage.windowsazure.com)).
2. View the list of **Service Bus** namespaces for your subscription.
3. Create a new **Service Bus** namespace by using the following values:
   * Namespace Name: **sb20532[Your Name]**
   * Region: **Select the region that is closest to you**
   * Type: **Messaging**
   * Messaging Tier: **Standard**

* **Note:** It takes approximately 1-2 minutes to create your Service Bus namespace instance.

1. Note down the connection string for your newly created Service Bus namespace.
2. Create a new **queue** in your namespace by using the following details:
   * **Queue** **Name**: signin
   * **Namespace**: sb20532[Your Name]

**Results:** After completing this exercise, you will have created a Service Bus namespace and queue by using the Portal.

## Exercise 2: Using Azure Queue Storage for Document Generation

### Scenario

In this exercise you will:

* Create queue messages by using Storage queues.
* Consume queue messages from Storage queues.

The main tasks for this exercise are as follows:

1. Update worker role to consume requests from the queue.
2. Update administration application to add requests to the queue.
3. Create a Storage account instance.
4. Generate the test data.
5. Debug and verify the application.

#### Task 1: Update worker role to consume requests from the queue

1. Open the **Contoso.Events** solution from the following location:

* File location: **Allfiles (F):\Mod08\Labfiles\Starter\Contoso.Events**

1. Open the **TableStorageQueueHelper.cs** file.
2. Add a using statement for the **System.Configuration** namespace to the top of the file:

* using System.Configuration;

1. In the **TableStorageQueueHelper** constructor, get the queue client and the name of the queue by using the following code:

* CloudStorageAccount storageAccount = base.StorageAccount;  
    
  \_queueClient = storageAccount.CreateCloudQueueClient();  
    
  \_signInQueueName = ConfigurationManager.AppSettings["SignInQueueName"];

1. Remove the existing code from the **Receive** method.
2. Create a new **CloudQueue** instance, and then ensure that the instance exists by using the following code:

* CloudQueue queue = \_queueClient.GetQueueReference(\_signInQueueName);  
    
  queue.CreateIfNotExists();

1. Get the *CloudQueueMessage* variable by calling the **GetMessage** method of the **CloudQueue** class, and then pass the message variable into the constructor of the **TableStorageQueueMessage** class, as shown in the following code:

* CloudQueueMessage message = queue.GetMessage();  
    
  return new TableStorageQueueMessage(message);

1. In the **CompleteMessage** method, create a new **CloudQueue** instance and ensure that the instance exists by using the following code:

* CloudQueue queue = \_queueClient.GetQueueReference(\_signInQueueName);  
    
  queue.CreateIfNotExists();

1. Call the **DeleteMessage** method of the **CloudQueue** class with the *CloudQueueMessage* variable as the first parameter, as shown in the following code:

* queue.DeleteMessage(message);

#### Task 2: Update administration application to add requests to the queue

1. Open the **SignInSheetViewModel.cs** file located in the **Contoso.Events.ViewModels** project.
2. In the **GenerateSignInSheetTableStorage** method, create a **CloudStorageAccount** instance and a **CloudQueueClient** instance with the **Microsoft.WindowsAzure.Storage.ConnectionString** setting value by using the following code:

* CloudStorageAccount storageAccount = CloudStorageAccount.Parse(tableStorageConnectionString);  
    
  CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

1. Using the **SignQueueName** cloud setting value, create a new **CloudQueue** instance and ensure that the instance exists, as shown in the following code:

* CloudQueue queue = queueClient.GetQueueReference(signInQueueName);  
    
  queue.CreateIfNotExists();

1. Add the message to the queue by creating a new instance of the **CloudQueueMessage** class using the string **message** as the constructor parameter, and then adding the new instance with the **CloudQueue AddMessage** method, as shown in the following code:

* CloudQueueMessage queueMessage = new CloudQueueMessage(message);  
    
  queue.AddMessage(queueMessage);

#### Task 3: Create a Storage Account Instance

1. Sign in to the Azure Portal (<https://portal.azure.com>).
2. Create a Storage Account using the following details:
   * Name: **provide a globally unique value**
   * Replication: **Locally Redundant Storage (LRS)**
   * Resource Group: **20532**
   * Location: **Select the region that is closest to your location**
3. Record the *account name* for your newly created Storage Account instance.
4. Select an access key and record the *connection string* for that key.

#### Task 4: Generate the test data

1. Locate the **Contoso.Events.Data.Generation** project.
2. Update the **app.config** file in the project to use your *Storage Account*'s connection string in the **StorageConnectionString** *AppSetting*.
3. Debug the **Contoso.Events.Data.Generation** project to generate the SQL and storage tables data.

#### Task 5: Debug and verify the application

1. Update your solution configuration to start the **Contoso.Events.Web**, **Contoso.Events.Worker** and **Contoso.Events.Management** projects when the solution is debugged.
2. Update the **web.config** file in the **Contoso.Events.Web** project to use your *Storage Account*'s connection string in the **Microsoft.WindowsAzure.Storage.ConnectionString** *AppSetting*.
3. Update the **web.config** file in the **Contoso.Events.Management** project to use your *Storage Account*'s connection string in the **Microsoft.WindowsAzure.Storage.ConnectionString** *AppSetting*.
4. Update the **app.config** file in the **Contoso.Events.Worker** project to use your *Storage Account*'s connection string in the following locations:
   * **StorageConnectionString** *AppSetting*
   * **AzureWebJobsStorage** *ConnectionString*
   * **AzureWebJobsDashboard** *ConnectionString*
5. Update the **app.config** file in the **Contoso.Events.Worker** project to use your *Service Bus*'s connection string in the following locations:
   * **AzureWebJobsServiceBus** *ConnectionString*
6. Debug the **Contoso.Events.Web**, **Contoso.Events.Worker** and **Contoso.Events.Management** projects.
7. In the **Contoso Events Administration** web application,
8. Switch to the **Contoso.Events – Microsoft Visual Studio** window.
9. Open the **Functions.cs** file in the **Contoso.Events.Worker** solution.
10. Add a breakpoint to the following line in the **ProcessQueueMessage** method:

* HandleMessage(message);

1. Switch to the **Home - Contoso.Events.Administration** browser window.
2. In the **Contoso Events Administration** web application, select any event, generate a sign-in sheet, and then download the sign-in sheet.
3. Verify that the application pauses execution at the debug breakpoint.
4. Stop debugging in Visual Studio, and then close Internet Explorer.

**Results:** After completing this exercise, you will have created and consumed messages from Storage queues.

## Exercise 3: Using Service Bus Queues for Document Generation

### Scenario

In this exercise, you will:

* Create queue messages by using Service Bus queues.
* Consume queue messages from Service Bus queues.

The main tasks for this exercise are as follows:

1. Update worker role to consume requests from the queue.
2. Update administration application to add requests to the queue.
3. Debug and verify the application.

#### Task 1: Update worker role to consume requests from the queue

1. Update the **app.config** file in the **Contoso.Events.Worker** project to use the connection string for your **Service Bus** namespace as the setting value for the **Microsoft.ServiceBus.ConnectionString** setting:
   * Name: **Microsoft.ServiceBus.ConnectionString**
   * Value: **[Your Service Bus Connection String]**
2. Open the **ServiceBusQueueHelper.cs** file in the **Contoso.Events.Worker** project.
3. Add a using statement for the **System.Configuration** namespace to the top of the file:

* using System.Configuration;

1. In the **ServiceBusQueueHelper** constructor, get the queue client by using the following code:

* string serviceBusConnectionString = ConfigurationManager.AppSettings["Microsoft.ServiceBus.ConnectionString"];  
    
  string signInQueueName = ConfigurationManager.AppSettings["SignInQueueName"];  
    
  \_client = QueueClient.CreateFromConnectionString(serviceBusConnectionString, signInQueueName);

1. Remove the existing code from the **Receive** method.
2. Invoke the **Receive** method of the **QueueClient** class, store the result in a **BrokeredMessage**, and then pass the new variable into the **ServiceBusQueueMessage** constructor, as shown in the following code:

* BrokeredMessage message = \_client.Receive();  
    
  return new ServiceBusQueueMessage(message);

1. In the **CompleteMessage** method, invoke the **Complete** method of the **BrokeredMessage** parameter by using the following code:

* message.Complete();

1. In the **AbandonMessage** method, invoke the **Abandon** method of the **BrokeredMessage** parameter by using the following code:

* message.Abandon();

1. Install Version **1.1.2** of the **Microsoft.Azure.WebJobs.ServiceBus** NuGet package in the **Contoso.Events.Worker** project.

* Install-Package Microsoft.Azure.WebJobs.ServiceBus -Version 1.1.2

1. Open the **Functions.cs** file in the **Contoso.Events.Worker** project.
2. Locate the method declaration for the **ProcessQueueMessage** method.
3. Update the parameter attribute of type **QueueTrigger** to type **ServiceBusTrigger**.
4. Open the **Program.cs** file in the **Contoso.Events.Worker** project.
5. In the **Main** method, create a new instance of the **JobHostConfiguration** class and then enable the **Service Bus** extension.

* JobHostConfiguration config = new JobHostConfiguration();  
   config.UseServiceBus();

1. In the **Main** method, update the initialization of the **JobHost** instance by passing in the **JobHostConfiguration** as a constructor parameter:

* var host = new JobHost(config);

#### Task 2: Update administration application to add requests to the queue

1. Update the **Web.config** file in the **Contoso.Events.Management** project to use the connection string for your **Service Bus** namespace as the setting value for the **Microsoft.ServiceBus.ConnectionString** app setting:
   * Key: **Microsoft.ServiceBus.ConnectionString**
   * Value: **[Your Service Bus Connection String]**
2. Open the **SignInSheetViewModel.cs** file.
3. In the constructor, locate the following line of code:

* GenerateSignInSheetTableStorage(context, eventItem, messageString);

1. Replace the above line of code with the following line of code:

* GenerateSignInSheetServiceBus(context, eventItem, message);

1. In the **GenerateSignInSheetServiceBus** method, create a **QueueClient** instance with the **Microsoft.WindowsAzure.Storage.ConnectionString** setting value and the **SignInQueueName** setting value by using the following code:

* QueueClient client = QueueClient.CreateFromConnectionString(serviceBusConnectionString, signInQueueName);

1. Add the message to the queue by creating a new instance of the **BrokeredMessage** class by using the **QueueMessage** message as the constructor parameter and adding the new instance with the **QueueClient** *Send* method, by using the following code:

* BrokeredMessage queueMessage = new BrokeredMessage(message);  
  client.Send(queueMessage);

#### Task 3: Debug and verify the application

1. Debug the solution.
2. In the **Contoso Events Administration** web application, select any event, generate a sign-in sheet, and then download the sign-in sheet.
3. Verify that the application pauses execution at the debug breakpoint.
4. Stop debugging in Visual Studio 2017, and then close Internet Explorer.

**Results:** After completing this exercise, you will have created and consumed messages from Service Bus Queues.

©2016 Microsoft Corporation. All rights reserved. The text in this document is available under the [Creative Commons Attribution 3.0 License](https://creativecommons.org/licenses/by/3.0/legalcode), additional terms may apply. All other content contained in this document (including, without limitation, trademarks, logos, images, etc.) are **not** included within the Creative Commons license grant. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.  
This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it. Some examples are for illustration only and are fictitious. No real association is intended or inferred. Microsoft makes no warranties, express or implied, with respect to the information provided here.