# 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

Currently, your on-premises Contoso Events application uses a WCF service to list the hotels that are near a location. You would like to continue to use the WCF service, but you cannot modify your company’s firewall. You also would not like to expose the true network location of the WCF service. You have decided to use Service Bus relays so that you have a common endpoint that you can provide to client applications. You will start by using that endpoint in your Contoso Events web application.

### Objectives

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

* Create an Azure Service Bus Namespace
* Create an Azure Storage Queue Instance
* Access Azure Storage Queues and Azure Service Bus using .NET

### Lab Setup

* *Estimated Time*: 90 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

#### Task 1: Sign in to the Azure Portal

1. Sign in to the Azure Portal (https://portal.azure.com).
2. If this is your first time logging in to the Azure portal, you will see a dialog with a tour of the portal. Click Get Started.

#### Task 2: Create Service Bus Namespace

1. Create a new **Service Bus** with the following details:
   * Name: bus20532[your name in lowercase here]
   * Pricing Tier: Basic
   * New Resource group: MOD08QBUS
   * Location: the region that is closest to your location

* **Note**: Wait for Azure to finish creating the Service Bus namespace prior to moving forward with the lab. You will receive a notification when the *Service Bus Namespace* is created.

1. Access the **Shared access policies** blade of your newly created **Service Bus** instance.
2. Create a new **SAS Policy** with the following details:
   * Policy name: ReadWriteOnly
   * Manage: Unchecked
   * Send: Checked
   * Listen: Checked

* **Note**: Wait for Azure to finish creating the policy prior to moving forward with the lab. You will receive a notification when the *policy* is created.

1. Access the newly created **ReadWriteOnly** policy and record the value of the **Primary Connection String** field. You will use this value later in this lab.
2. Access the Queues blade and create a new queue named **documentqueue** with the default settings.

In the *Service Bus Namespace* blade, locate the **Entities** section on the left-side of the blade and click the **Queues** link.

**Note**: Wait for Azure to finish creating the queue prior to moving forward with the lab. You will receive a notification when the *queue* is created.

#### Task 3: Create Other Azure Assets

1. Create a new **Storage Account** with the following details:
   * Name: star20532[your name in lowercase here]
   * Deployment model: Resource manager
   * Account kind: StorageV2 (general purpose v2)
   * Location: Closest region to your current location
   * Replication: Locally-redundant storage (LRS)
   * Performance: Standard
   * Access tier: Hot
   * Secure transfer required: Enabled
   * Resource group: MOD08QBUS
   * Virtual networks: Disabled

* **Note**: Wait for Azure to finish creating the storage account prior to moving forward with the lab. You will receive a notification when the *Storage Account* is created.

1. Access the **Access Keys** blade of your newly created **Storage account** instance.
2. Record any one of the **Keys** and record the value in the **Connection string** field. You will use this value later in this lab.
3. Create a new **SQL Database** with the following details:
   * Name: db20532
   * Resource Group: MOD08QBUS
   * Select Source: Blank Database
   * Server Name: sv20532[*Your Name Here*]
   * Admin login: testuser
   * Password/Confirm Password: TestPa$$w0rd
   * Location: Closest to your location
   * Pricing Tier: Basic

* **Note**: Wait for Azure to finish creating the SQL Database instance prior to moving forward with the lab. You will receive a notification when the *SQL Database* is created.

1. Access the **Connection strings** blade of your newly created **SQL database**.
2. In the **Connection strings** pane, copy the value of the **ADO.NET** connection string. Be sure to replace the placeholder values for {your\_username} and {your\_password} with the values **testuser** and **TestPa$$w0rd** respectively.

* **Note**: For example, if your copied connection string is Server=tcp:sv20532microsoft.database.windows.net,1433;Initial Catalog=db20532;Persist Security Info=False;User ID={your\_username};Password={your\_password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;, your updated connection string would be Server=tcp:sv20532microsoft.database.windows.net,1433;Initial Catalog=db20532;Persist Security Info=False;User ID=testuser;Password=TestPa$$w0rd;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;

1. Access the **Firewalls and virtual networks** blade of your newly created **SQL server** instance.
2. Add your virtual machine's IP Address to the list of allowed IP Address ranges.

* **Note**: It might take couple of minutes for the firewall changes to get updated on server.

1. Create a new **Azure Cosmos DB** instance with the following details:
   * ID: nosql20532[your name in lowercase here]
   * API: SQL
   * Resource Group: MOD08QBUS
   * Location: Closest to your location
   * Enable geo-redundancy: Unchecked
   * Configure virtual networks: Disabled

* **Note**: Wait for Azure to finish creating the Azure Cosmos DB account prior to moving forward with the lab. You will receive a notification when the *Azure Cosmos DB account* is created.

1. Access the **Keys** blade of your newly created **Azure Cosmos DB** instance.
2. Record the value of the **URI** and **PRIMARY KEY**. You will use these values later in this lab.

#### Task 4: Update Settings in Contoso.Events Projects

1. Open the **Contoso.Events.sln** solution found in **Allfiles (F):\Mod08\Labfiles\Starter**.
2. Build the Solution.
3. Open **appsettings.json** located in the **Contoso.Events.Management** project.
4. Locate and update the value of the following properties:
   * StorageSettings.ConnectionString: [Storage Account Connection String]
   * ServiceBusSettings.ConnectionString: [Service Bus Connection String]
   * CosmosSettings.EndpointUrl: [Azure Cosmos DB URI]
   * CosmosSettings.AuthorizationKey: [Azure Cosmos DB Key]
   * ConnectionStrings.EventsContextConnectionString: [SQL Database Connection String]
5. Save the **appsettings.json** file.
6. Open **local.settings.json** located in the **Contoso.Events.Worker** project.
7. Locate and update the value of the following properties:
   * AzureWebJobsStorage: [Storage Account Connection String]
   * AzureWebJobsDashboard: [Storage Account Connection String]
   * EventsContextConnectionString: [SQL Database Connection String]
   * ServiceBusConnectionString: [Service Bus Connection String]
   * CosmosEndpointUrl: [Azure Cosmos DB URI]
   * CosmosAuthorizationKey: [Azure Cosmos DB Key]
8. Save the **local.settings.json** file.

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

#### Task 1: Implement QueueContext Class using the IQueueContext Interface

1. Open **QueueContext.cs** located in the **Contoso.Events.Data** project.
2. Locate and replace the **SendQueueMessageAsync** method with the following method implementation:

* public async Task SendQueueMessageAsync(string eventKey)  
   {  
   CloudStorageAccount account = CloudStorageAccount.Parse(StorageSettings.ConnectionString);  
   CloudQueueClient queueClient = account.CreateCloudQueueClient();  
   CloudQueue queue = queueClient.GetQueueReference(StorageSettings.QueueName);  
   await queue.CreateIfNotExistsAsync();  
    
   CloudQueueMessage message = new CloudQueueMessage(eventKey);  
   await queue.AddMessageAsync(message);  
   }

1. Save the **QueueContext.cs** file.

#### Task 2: Register QueueContext Implementation with ASP.NET MVC Dependency Injection

1. Open **Startup.cs** located in the **Contoso.Events.Management** project.
2. Locate and replace the **ConfigureServices** method with the following method implementation:

* public void ConfigureServices(IServiceCollection services)  
   {  
   services.AddRouting(options => options.LowercaseUrls = true);  
    
   services.AddMvc();  
    
   services.Configure<ApplicationSettings>(Configuration.GetSection(nameof(ApplicationSettings)));  
   services.Configure<CosmosSettings>(Configuration.GetSection(nameof(CosmosSettings)));  
   services.Configure<StorageSettings>(Configuration.GetSection(nameof(StorageSettings)));  
   services.Configure<ServiceBusSettings>(Configuration.GetSection(nameof(ServiceBusSettings)));  
    
   services.AddSingleton<IConfiguration>(Configuration);  
   services.AddSingleton<RegistrationContext>();  
    
   services.AddTransient<BlobContext>();  
   services.AddTransient<IQueueContext, QueueContext>();  
    
   services.AddDbContext<EventsContext>(options => options.UseSqlServer(Configuration.GetConnectionString("EventsContextConnectionString")));  
   }

1. Save the **Startup.cs** file.

#### Task 3: Implement Storage Queue Trigger for Azure Function

1. Open **ProcessDocuments.cs** located in the **Contoso.Events.Worker** project.
2. Locate and update the **Run** method signature with the following method signature:

* public static async Task Run([QueueTrigger("signinqueue", Connection = "AzureWebJobsStorage")]string message, [Blob("signinsheets/{queueTrigger}.docx", FileAccess.Write)] Stream blob, TraceWriter log)

1. Save the **ProcessDocuments.cs** file.

#### Task 4: Validate Sign-In Sheet Generation

1. Access the **Properties** of the solution.
2. Adjust **Startup Project** settings as listed:
   * Startup: Multiple startup projects
   * Contoso.Events.Managment: Start
   * Contoso.Events.Worker: Start
3. Debug the solution.
4. On the home page of the web application, verify that it displays a list of events.
5. Generate a sign in sheet for any of the events in the list. Remember the name of the event as you will click the same button again in this lab.
6. You will observe that the Azure Function will immediately detect the new document in the **signinqueue** queue and begin processing that document. Once it is done processing, refresh the browser window.
7. Download and Open the generated sign in sheet file on your local machine. Observe and then close the file.
8. Close both the browser and console window to return to visual studio.

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

#### Task 1: Implement ServiceBusContext Class using the IQueueContext Interface

1. Open **ServiceBusContext.cs** located in the **Contoso.Events.Data** project.
2. Locate and replace the **SendQueueMessageAsync** method with the following method implementation:

* public async Task SendQueueMessageAsync(string eventKey)  
   {  
   QueueClient queueClient = new QueueClient(ServiceBusSettings.ConnectionString, ServiceBusSettings.QueueName);  
   byte[] messageContent = Encoding.UTF8.GetBytes(eventKey);  
   Message message = new Message(messageContent);  
   message.Label = eventKey;  
   await queueClient.SendAsync(message);  
   }

1. Save the **ServiceBusContext.cs** file.

#### Task 2: Register ServiceBusContext Implementation with ASP.NET MVC Dependency Injection

1. Open **Startup.cs** located in the **Contoso.Events.Management** project.
2. Locate and replace the **ConfigureServices** method with the following method implementation:

* public void ConfigureServices(IServiceCollection services)  
   {  
   services.AddRouting(options => options.LowercaseUrls = true);  
    
   services.AddMvc();  
    
   services.Configure<ApplicationSettings>(Configuration.GetSection(nameof(ApplicationSettings)));  
   services.Configure<CosmosSettings>(Configuration.GetSection(nameof(CosmosSettings)));  
   services.Configure<StorageSettings>(Configuration.GetSection(nameof(StorageSettings)));  
   services.Configure<ServiceBusSettings>(Configuration.GetSection(nameof(ServiceBusSettings)));  
    
   services.AddSingleton<IConfiguration>(Configuration);  
   services.AddSingleton<RegistrationContext>();  
    
   services.AddTransient<BlobContext>();  
   services.AddTransient<IQueueContext, ServiceBusContext>();  
    
   services.AddDbContext<EventsContext>(options => options.UseSqlServer(Configuration.GetConnectionString("EventsContextConnectionString")));  
   }

1. Save the **Startup.cs** file.

#### Task 3: Implement Service Bus Queue Trigger for Azure Function

1. Open **ProcessDocuments.cs** located in the **Contoso.Events.Worker** project.
2. Locate and update the **Run** method signature with the following method signature:

* public static async Task Run([ServiceBusTrigger("documentqueue", Connection = "ServiceBusConnectionString")]string message, [Blob("signinsheets/{label}.docx", FileAccess.Write)] Stream blob, TraceWriter log)

1. Save the **ProcessDocuments.cs** file.

#### Task 4: Validate Sign-In Sheet Generation

1. Debug the solution.
2. On the home page of the web application, verify that it displays a list of events.
3. Generate a sign in sheet for any of the events in the list. Remember the name of the event as you will click the same button again in this lab.
4. You will observe that the Azure Function will immediately detect the new document in the **signinqueue** queue and begin processing that document. Once it is done processing, refresh the browser window.
5. Download and Open the generated sign in sheet file on your local machine. Observe and then close the file.
6. Close both the browser and console window to return to visual studio.

## Exercise 4: Cleanup Subscription

#### Task 1: Open Cloud Shell

1. At the top of the portal, click the **Cloud Shell** icon to open a new shell instance.
2. In the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press **Enter** to list all resource groups in the subscription:

* az group list

1. Type in the following command and press **Enter** to view a list of possible CLI commands to *delete a Resource Group*:

* az group delete --help

#### Task 2: Delete Resource Group

1. Type in the following command and press **Enter** to delete the **MOD08QBUS** *Resource Group*:

* az group delete --name MOD08QBUS --no-wait --yes

1. Close the **Cloud Shell** prompt at the bottom of the portal.

#### Task 3: Close Active Applications

1. Close the currently running web browser application.
2. Close the currently running **Visual Studio** application.

**Review**: In this exercise, you "cleaned up your subscription" by removing the **Resource Groups** used in this lab.d

©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.