# Module 7: Storing and Consuming Files from Azure Storage

# Lab: Storing Generated Documents in Azure Storage Blobs

### Scenario

You need a place to store the Word documents that are generated by the Contoso Events application. You decide store the generated Word documents in blobs. You also decide to create a protected container so that the Word documents are not accessed by anonymous users. Finally, you want to create the logic to generate SAS tokens for temporary access to one of the Word documents.

### Objectives

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

* Access Azure Storage Blobs using .NET
* Populating Azure Storage with Containers and Media
* Implementing Direct and Indirect Blob Access using .NET
* Generating SAS Tokens for Azure Storage Blobs

### 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: Implementing Azure Storage Blobs

#### 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 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
   * New resource group: MOD07STOR
   * 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: MOD07STOR
   * 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: MOD07STOR
   * 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 3: Update Settings in Contoso.Events Projects

1. Open the **Contoso.Events.sln** solution found in **Allfiles (F):\Mod07\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]
   * 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]
   * CosmosEndpointUrl: [Azure Cosmos DB URI]
   * CosmosAuthorizationKey: [Azure Cosmos DB Key]
8. Save the **local.settings.json** file.

## Exercise 2: Populating the Container with Files and Media

#### Task 1: Implement Blob Trigger and Output For Azure Functions

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

* public static async Task Run([BlobTrigger("signinsheets-pending/{name}")] Stream input, string name, [Blob("signinsheets/{name}", FileAccess.Write)] Stream output, TraceWriter log)  
   {  
   log.Info($"Request received to generate sign-in sheet for event: {name}");  
    
   string eventKey = Path.GetFileNameWithoutExtension(name);  
   using (MemoryStream stream = await ProcessStorageMessage(eventKey))  
   {  
   byte[] byteArray = stream.ToArray();  
   await output.WriteAsync(byteArray, 0, byteArray.Length);  
   }  
    
   log.Info($"Request received to generate sign-in sheet for event: {name}");  
   }

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

#### Task 2: Implement Blob Upload in BlobContext Class

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

* public async Task<ICloudBlob> UploadBlobAsync(string blobName, Stream stream)  
   {  
   CloudStorageAccount account = CloudStorageAccount.Parse(StorageSettings.ConnectionString);  
   CloudBlobClient blobClient = account.CreateCloudBlobClient();  
   CloudBlobContainer container = blobClient.GetContainerReference($"{StorageSettings.ContainerName}-pending");  
   await container.CreateIfNotExistsAsync();  
    
   ICloudBlob blob = container.GetBlockBlobReference(blobName);  
   stream.Seek(0, SeekOrigin.Begin);  
   await blob.UploadFromStreamAsync(stream);  
    
   return new DownloadPayload { Stream = blobStream, ContentType = blob.Properties.ContentType };  
   }

#### Task 3: 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 **signinsheets-pending** container 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.

#### Task 5: Validate Azure Storage Data

1. Return to the **Azure Portal**
2. Access the **Containers** blade of the **storage account** instance created perviously in the lab.
3. Observe the documents in the **signinsheets** link.

## Exercise 3: Retrieving Files and Media from the Container

#### Task 1: Implement Blob Access using Direct Download

1. Open **BlobContext.cs** located in the **Contoso.Events.Data** project.
2. Locate and replace the **GetStreamAsync** method with the following method implementation:
3. In the code editor tab for the **BlobContext.cs** file, locate the **BlobContext** class:

* public async Task<DownloadPayload> GetStreamAsync(string blobId)  
   {  
   CloudStorageAccount account = CloudStorageAccount.Parse(StorageSettings.ConnectionString);  
   CloudBlobClient blobClient = account.CreateCloudBlobClient();  
   CloudBlobContainer container = blobClient.GetContainerReference(StorageSettings.ContainerName);  
   await container.CreateIfNotExistsAsync();  
    
   ICloudBlob blob = container.GetBlockBlobReference(blobId);  
   Stream blobStream = await blob.OpenReadAsync(null, null, null);  
    
   return blobStream;  
   }

#### Task 2: Validate Sign-In Sheet Download

1. Set the **Contoso.Events.Management** project as the Startup Project.
2. Debug the solution.
3. On the home page of the web application, verify that it displays a list of events.
4. Click the **Generate Sign-In Sheet** button for the same event you used earlier in this lab.
5. Click the **Download Sign-In Sheet Stream** link to download the document as a stream from the web server
6. Close the browser window that is displaying the website.
7. In the **Solution Explorer** pane, right-click the **Contoso.Events.Management** project, and then click **Set as Startup Project**.
8. On the **Debug** menu, click **Start Debugging**.
9. On the home page of the web application, verify that it displays a list of events.
10. Click the **Generate Sign-In Sheet** button for the same event you used earlier in this lab.
11. On the sign-in sheet page, you should see a section titled **Sign-In Document Already Exists**.
12. Click the **Download Sign-In Sheet Stream** link to download the document as a stream from the web server.
13. Close the browser window that is displaying the website.

## Exercise 4: Specifying Permissions for the Container

#### Task 1: Securing Blob Container

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

* public async Task<string> GetSecureUrlAsync(string blobId)  
   {  
   CloudStorageAccount account = CloudStorageAccount.Parse(StorageSettings.ConnectionString);  
   CloudBlobClient blobClient = account.CreateCloudBlobClient();  
   CloudBlobContainer container = blobClient.GetContainerReference(StorageSettings.ContainerName);  
   await container.CreateIfNotExistsAsync();  
    
   SharedAccessBlobPolicy blobPolicy = new SharedAccessBlobPolicy  
   {  
   SharedAccessExpiryTime = DateTime.Now.AddHours(0.25d),  
   Permissions = SharedAccessBlobPermissions.Read  
   };  
    
   BlobContainerPermissions blobPermissions = new BlobContainerPermissions  
   {  
   PublicAccess = BlobContainerPublicAccessType.Off  
   };  
   blobPermissions.SharedAccessPolicies.Add("ReadBlobPolicy", blobPolicy);  
    
   await container.SetPermissionsAsync(blobPermissions);  
    
   string sasToken = container.GetSharedAccessSignature(new SharedAccessBlobPolicy(), "ReadBlobPolicy");  
    
   ICloudBlob blob = container.GetBlockBlobReference(blobId);  
   Uri blobUrl = blob.Uri;  
    
   return blobUrl.AbsoluteUri + sasToken;  
   }

#### Task 3: Validate Sign-In Sheet Hyperlink

1. Debug the solution.
2. On the home page of the web application, verify that it displays a list of events.
3. Click the **Generate Sign-In Sheet** button for the same event you used earlier in this lab.
4. Click the **Use Sign-In Sheet Hyperlink** link to download the document.
5. Close the browser window that is displaying the website.

## Exercise 5: 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 **MOD07STOR** *Resource Group*:

* az group delete --name MOD07STOR --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.b

©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.