# 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:

* Create a container by using the Azure Portal.
* Add blobs to the container by using the Azure Storage SDK.
* Download blobs from the container by using **MemoryStream** objects.
* Generate SAS tokens for accessing a blob.

### 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: Implementing Azure Storage Blobs

### Scenario

In this exercise, you will:

* Create a blob container by using the Portal.
* Add blob files to the container.
* Access the blob files in the container.

The main tasks for this exercise are as follows:

1. Sign in to the Azure Portal.
2. Create a container by using the Portal.
3. Add and access blob files in your container.

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

Sign in to the Azure Portal (<https://portal.azure.com>).

#### Task 2: Create a container by using the Portal

1. View the list of Storage instances for your subscription.
2. Go to the **Blob Service** blade for your **stor20532[*Your Name*]** subscription.
3. Create a new Container with the following values:
   * Name: **example**
   * Access Type: **Container**
4. Record the *account name* for your Storage Account instance.
5. Select an access key and record the *connection string* for that key.

#### Task 3: Add and access blob files in your container

1. Open Visual Studio 2017.
2. Open the **Storage Accounts** node in the **Cloud Explorer** pane.
3. Open the **example** container in your previously modified storage account:
   * Account name: **stor20532*[Your Name]***
   * Container name: **example**
4. In the **Container** tab, upload the sample file that is located at the following location:
   * Sample File Location: **(F):\Mod07\LabFiles\Starter\samplefile.txt**
5. View the sample file’s text in Internet Explorer by using the following URL:
   * <https://[account].blob.core.windows.net/example/samplefile.txt>

**Results:** After completing this exercise, you will have created a blob container by using the Portal and viewed the blobs in the container.

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

### Scenario

In this exercise, you will:

* Use the Azure Storage SDK to access blobs in your storage account.
* Use the Azure Storage SDK to create blobs in your storage account.

The main tasks for this exercise are as follows:

1. Open the blob helper in the Web App worker project.
2. Add Word documents to the container after they are created.

#### Task 1: Open the blob helper in the Web App worker project

1. Open the **Contoso.Events** solution from the following location:
   * File location: **Allfiles (F):\Mod07\Labfiles\Starter\Contoso.Events**
2. Open the **BlobStorageHelper.cs** file in the **Contoso.Events.Worker** project.

#### Task 2: Add Word documents to the container after they are created

1. Remove the existing code from the **CreateBlob** method.
2. Create a *CloudBlobContainer* variable for the **signin** container and ensure that the container exists by using the following code:

* CloudBlobContainer container = \_blobClient.GetContainerReference("signin");  
    
  container.CreateIfNotExists();

1. Create a name for your blob by using the following format: **“{eventKey}\_SignIn\_{date}”**.
2. Create a blob reference by using the new **blobName** variable, use the Seek method on the **MemoryStream** to set the position back to the beginning (Origin), and then upload the stream to the blob by using the **UploadFromStream** method as shown in the following code:

* ICloudBlob blob = container.GetBlockBlobReference(blobName);  
    
  stream.Seek(0, SeekOrigin.Begin);  
    
  blob.UploadFromStream(stream);

1. Return the blob’s **Uri** property as the result of the method.

**Results:** After completing this exercise, you will have used the Azure Storage SDK to manage blobs and containers in your storage account.

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

### Scenario

In this exercise, you will:

* Retrieve blobs from your storage account by using the Azure Storage SDK.

The main tasks for this exercise are as follows:

1. Download documents from blob storage and stream to the client.
2. Generate the Test Data.
3. Download generated sign-in sheets from the blob storage.

#### Task 1: Download documents from blob storage and stream to the client

1. Open the **DownloadViewModel.cs** file in the **Contoso.Events.ViewModels** project.
2. Remove the existing code from the **GetStream** method.
3. Use the *\_storageAccount* variable to create a new **CloudBlobClient** instance, create a *CloudBlobContainer* variable for the **signin** container, and then ensure that the container exists as shown in the following code:

* CloudBlobClient blobClient = \_storageAccount.CreateCloudBlobClient();  
    
  CloudBlobContainer container = blobClient.GetContainerReference("signin");  
    
  container.CreateIfNotExists();

1. Create a blob reference by using the new **\_blobId** variable, and then use the **OpenReadAsync** method to create a *Stream* variable as shown in the following code:

* ICloudBlob blob = container.GetBlockBlobReference(\_blobId);  
    
  Stream blobStream = await blob.OpenReadAsync();

1. Return a new instance of the **DownloadPayload** class by assigning the *Stream* variable to the **DownloadPayload.Stream** property and assign the *ICloudBlob* variable’s **Properties.ContentType** value to the **DownloadPayload.ContentType** property.

#### Task 2: 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 3: Download generated sign-in sheets from the blob storage

1. Update your solution configuration to start both 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. Debug the **Contoso.Events.Web**, **Contoso.Events.Worker** and **Contoso.Events.Management** projects.
6. In the **Contoso Events Administration** web application, select any event, generate a sign-in sheet, and then download the sign-in sheet.

* **Note:** Currently, the web application gets a Stream of the blob and uses MVC’s file helper to return the file to the user. The implementation is available to view in the **DownloadSignIn** method of the **HomeController** class in the **Contoso.Events.Management** project.

**Results:** After completing this exercise, you will have downloaded blobs from your storage account by using the Azure Storage SDK.

## Exercise 4: Specifying Permissions for the Container

### Scenario

In this exercise, you will:

* Use the Visual Studio 2017 Cloud Explorer to modify a container.
* Generate a SAS token by using the Azure Storage SDK.

The main tasks for this exercise are as follows:

1. Modify Container Access using Cloud Explorer.
2. Generate temporary SAS tokens by using the SDK.
3. Download documents from a protected container by using the SAS token.

#### Task 1: Modify Container Access using Cloud Explorer

1. Switch to the *Contoso.Events - Visual Studio 2017* window, and then stop debugging.
2. Open the **Storage Accounts** node of the **Cloud Explorer** pane.
3. View the **signin** container in your recently used storage account:
   * Account name: **stor20532[*Your Name*]**
   * Container name: **signin**
4. Ensure that the value of the **Public Read Access** property is set to **Off**.

#### Task 2: Generate temporary SAS tokens by using the SDK

1. Open the **DownloadViewModel.cs** file in the **Contoso.Events.ViewModels** project.
2. Remove the existing code from the **GetSecureUrl** method.
3. Use the *\_storageAccount* variable to create a new **CloudBlobClient**, create a *CloudBlobContainer* variable for the **signin** container, and then ensure that the container exists as shown in the following code:

* CloudBlobClient blobClient = \_storageAccount.CreateCloudBlobClient();  
    
  CloudBlobContainer container = blobClient.GetContainerReference("signin");  
    
  container.CreateIfNotExists();

1. Create a new instance of the **SharedAccessBlobPolicy** class, set the expiry time to 15 minutes from the current time, and then set the blob permission to read as shown in the following code:

* SharedAccessBlobPolicy blobPolicy = new SharedAccessBlobPolicy();  
    
  blobPolicy.SharedAccessExpiryTime = DateTime.Now.AddHours(0.25d);  
    
  blobPolicy.Permissions = SharedAccessBlobPermissions.Read;

1. Create a new instance of the **BlobContainerPermissions** class, add the newly created **SharedAccessBlobPolicy** with the name **“ReadBlobPolicy”**, and then disable public access as shown in the following code:

* BlobContainerPermissions blobPermissions = new BlobContainerPermissions();  
    
  blobPermissions.SharedAccessPolicies.Add("ReadBlobPolicy", blobPolicy);  
    
  blobPermissions.PublicAccess = BlobContainerPublicAccessType.Off;

1. Use the **CloudBlobContainer** asynchronous **SetPermissionsAsync** method to apply the *BlobContainerPermissions* variable, and then generate a SAS token by using the **GetSharedAccessSignature** method as shown in the following code:

* await container.SetPermissionsAsync(blobPermissions);  
    
  string sasToken = container.GetSharedAccessSignature(new SharedAccessBlobPolicy(), "ReadBlobPolicy");

1. Create a blob reference by using the *\_blobId* variable, and then store the Blob’s Uri in a *Uri* variable as shown in the following code:

* ICloudBlob blob = container.GetBlockBlobReference(\_blobId);  
    
  Uri blobUrl = blob.Uri;

1. Concatenate the *Uri* variable’s **AbsoluteUri** property with the **sasToken**, and then return the new string as the result of the method.

#### Task 3: Download documents from a protected container by using the SAS token

1. Debug the solution again.
2. In the **Contoso Events Administration** web application, select any event, generate a sign-in sheet, and then download the sign-in sheet using the hyperlink instead.

* **Note:** You can always look at Internet Explorer’s download manager if you want to see the full URL that is used (with SAS Token) to download the file.

1. Close the **Internet Explorer** and **Contoso.Events – Visual Studio** windows.

**Results:** After completing this exercise, you will have modified the permissions of the containers and generated SAS tokens for the containers.

©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.