# Module 7: Storing and Consuming Files from Azure Storage

# Lab: Storing Generated Documents in Azure Storage Blobs

## Exercise 1: Implementing Azure Storage Blobs

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

1. On the Start screen, click the **Internet Explorer** tile.
2. Go to *(*[*https://portal.azure.com*](https://portal.azure.com/)*)*.
3. In the email address box, type the email address of your Microsoft account.
4. Click **Next**.
5. In the password box, type the password for your Microsoft account.
6. Click **Sign In**.
7. **Note**: If this is your first time logging in to the Portal, you may be prompted with a “Welcome” dialog. You can safely close this dialog and continue.

#### Task 2: Create The Storage Account

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **Storage Accounts**.
3. In the **Storage accounts** blade that displays, view your list of Storage instances.
4. At the top of the **Storage accounts** blade, click the **Add** button.
5. In the **Create storage account** blade that displays, perform the following steps:

In the **Name** field, provide the value **star20532[your name in lowercase here]**.

In the **Deployment model** section, ensure that the *Resource manager* option is selected.

In the **Account kind** list, ensure that the *StorageV2 (general purpose v2)* option is selected.

In the **Location** list, select the region closest to your current location.

Click on the **Replication** list and select the **Locally-redundant storage (LRS)** option.

In the **Performance** section, ensure that the *Standard* option is selected.

In the **Access tier** section, ensure that the *Hot* option is selected.

In the **Secure transfer required** section, ensure that the *Enabled* option is selected.

In the **Resource group** section, select the **Create new** option.

In the **Resource group** section, locate the dialog box and enter the value **MOD07STOR**.

In the **Virtual networks** section, ensure that the *Disabled* option is selected.

Click the **Create** button.

**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.

In the navigation pane on the left side of the Azure Portal, click **All services**.

In the **All services** blade that displays, click **Storage Accounts**.

In the **Storage accounts** blade that displays, select the Storage account instance that has a prefix of **star20532**.

In the *Storage account* blade, locate the **Settings** section on the left-side of the blade and click the **Access keys** link.

In the **Keys** pane, select any one of the keys and record the value in the **Connection string** field. You will use this value later in this lab.

In the navigation pane on the left side, click **All services**.

#### Task 3: Create the SQA AzureDatabase

1. In the **All services** blade that displays, click **SQL databases**.
2. In the **SQL databases** blade that displays, view your list of SQL databases.
3. At the top of the **SQL databases** blade, click the **Add** button.
4. In the *SQL database* blade that displays, locate the **Database name** field and provide the value **db20532**.
5. Locate the **Resource group** section, select the **Use existing** option and then select the option **MOD07STOR** from the list.
6. In the **Select source** list, select the value **Blank database**.
7. Click the **Server** option.
8. In the **Server** blade that displays, perform the following actions:
   1. Click **Create a new server**.
9. In the **New Server** blade that displays, locate the **Server Name** field.
10. In the **Server name** field, type **sv20532[*Your Name Here*]**.
11. In the **Server admin login** field, type **testuser** as login.
12. In the **Password** field, type **TestPa$$w0rd** as password.
13. In the **Confirm password** field, type **TestPa$$w0rd**.
14. In the **Location** list, select the region that is closest to your location.
15. Click the **Select** button.
16. Back in the **SQL database** blade, click the **Pricing Tier** option.
17. In the **Resource Configuration & Pricing** blade that displays, select the **Basic** option.
18. Click **Apply** to close the blade.
19. Back in the **SQL database** blade, click the **Create** button to create the SQL database and server.

**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.

In the navigation pane on the left side of the Azure Portal, click **All services**.

In the **All services** blade that displays, click **SQL Database**.

In the **SQL Database** blade that displays, select the SQL database instance named **db20532**.

In the *SQL database* blade, locate the **Settings** section on the left-side of the blade and click the **Connection strings** link.

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;

In the navigation pane on the left side of the Azure Portal, click **All services**.

In the **All services** blade that displays, click **SQL servers**.

In the **SQL servers** blade that displays, select the SQL database instance that has a prefix of **sv20532**.

In the *SQL server* blade, locate the **Security** section on the left-side of the blade and click the **Firewalls and virtual networks** link.

In the **Firewalls and virtual networks** pane, click the **Add client IP** button to add your virtual machine's IP Address to the list of allowed IP Address ranges.

Click on **Save** at the top of the blade. Once saved, close the confirmation dialog by clicking the **Ok** button.

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

#### Task 4: Create the CosmosDB database

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **Azure Cosmos DB**.
3. In the **Azure Cosmos DB** blade that displays, view your list of Storage instances.
4. At the top of the **Azure Cosmos DB** blade, click the **Add** button.
5. In the **New account** blade that displays, perform the following steps:

In the **ID** field, provide the value **nosql20532[your name in lowercase here]**.

In the **API** list, select the **SQL** option.

In the **Resource group** section, select the **Use existing** option.

In the **Resource group** section, select the value **MOD07STOR** from the list.

In the **Location** list, select the region closest to your current location.

Ensure that the *Enable geo-redundancy* checkbox is un-selected.

In the **Virtual networks** section, ensure that the *Disabled* option is selected.

Click the **Create** button.

**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.

In the navigation pane on the left side of the Azure Portal, click **All services**.

In the **All services** blade that displays, click **Azure Cosmos DB**.

In the **Azure Cosmos DB** blade that displays, select the Azure Cosmos DB account instance that has a prefix of **nosql20532**.

In the *Azure Cosmos DB account* blade, locate the **Settings** section on the left-side of the blade and click the **Keys** link.

In the **Keys** pane, record the values in the **URI** and **PRIMARY KEY** fields. You will use these values later in this lab.

#### Task 5: Update Settings in Contoso.Events Projects

1. On the Start screen, click the **Desktop** tile.
2. On the taskbar, click the **File Explorer** icon.
3. In the *This PC* window, go to **Allfiles (F):\Mod07\Labfiles\Starter**, and then double-click **Contoso.Events.sln**.
4. On the **Build** menu, click **Build Solution**.
5. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Management** project.
6. In the **Solution Explorer** pane, double-click **appsettings.json**.
7. In the JSON object, in line 16, locate the **ConnectionStrings.EventsContextConnectionString** path. Observe that the current value is empty:

"ConnectionStrings": {"EventsContextConnectionString": ""},

1. Update the value of the **EventsContextConnectionString** property by setting it's value to the **Connection String** of the **SQL Database** you recorded earlier in this lab.
2. In the JSON object, in line 12, locate the **CosmosSettings.EndpointUrl** path. Observe that the current value is empty:

"CosmosSettings": {"DatabaseId": "EventsDatabase","CollectionId": "RegistrationCollection","EndpointUrl": "","AuthorizationKey": ""},

1. Update the value of the **EndpointUrl** property by setting it's value to the **Endpoint Uri** of the **Azure Cosmos DB account** you recorded earlier in this lab.
2. In the JSON object, in line 13, locate the **CosmosSettings.AuthorizationKey** path. Observe that the current value is empty:

"CosmosSettings": {"DatabaseId": "EventsDatabase","CollectionId": "RegistrationCollection","EndpointUrl": "","AuthorizationKey": ""},

1. Update the value of the **AuthorizationKey** property by setting it's value to the **Key** of the **Azure Cosmos DB account** you recorded earlier in this lab.
2. In the JSON object, in line 6, locate the **StorageSettings.ConnectionString** path. Observe that the current value is empty:

"StorageSettings": { "ConnectionString": "", "ContainerName": "signinsheets"},

1. Update the value of the **ConnectionString** property by setting it's value to the **Connection String** of the **Storage account** you recorded earlier in this lab.
2. Save the **appsettings.json** file.
3. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Worker** project.
4. In the **Solution Explorer** pane, double-click **local.settings.json**.
5. In the JSON object, in line 4, locate the **AzureWebJobsStorage** path. Observe that the current value is empty:

"AzureWebJobsStorage": "",

1. Update the value of the **AzureWebJobsStorage** property by setting it's value to the **Connection String** of the **Storage account** you recorded earlier in this lab.
2. In the JSON object, in line 5, locate the **AzureWebJobsDashboard** path. Observe that the current value is empty:

"AzureWebJobsDashboard": "",

1. Update the value of the **AzureWebJobsDashboard** property by setting it's value to the **Connection String** of the **Storage account** you recorded earlier in this lab.
2. In the JSON object, in line 6, locate the **EventsContextConnectionString** path. Observe that the current value is empty:

"EventsContextConnectionString": "",

1. Update the value of the **EventsContextConnectionString** property by setting it's value to the **Connection String** of the **SQL Database** you recorded earlier in this lab.
2. In the JSON object, in line 7, locate the **CosmosEndpointUrl** path. Observe that the current value is empty:

"CosmosEndpointUrl": "",

1. Update the value of the **CosmosEndpointUrl** property by setting it's value to the **Endpoint Uri** of the **Azure Cosmos DB account** you recorded earlier in this lab.
2. In the JSON object, in line 8, locate the **CosmosAuthorizationKey** path. Observe that the current value is empty:

"CosmosAuthorizationKey": "",

1. Update the value of the **CosmosAuthorizationKey** property by setting it's value to the **Key** of the **Azure Cosmos DB account** you recorded earlier in this lab.
2. 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. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Worker** project and double-click the **ProcessDocuments.cs** file.
2. In the code editor tab for the **ProcessDocuments.cs** file, locate the **ProcessDocuments** class:

public static class ProcessDocuments

Within the **ProcessDocuments** class, locate the **Run** method:

public static async Task Run(Stream input, string name, Stream output, TraceWriter log)

Update the method signature of the **Run** method by adding a **BlobTrigger** parameter attribute to the **input** parameter specifying to match on any blob in the **signinsheets-pending** container and secondly by adding a **Blob** parameter attribute to the **output** parameter specifying to create a new blob in the **signinsheets** container with the same name as the blob that triggered the function execution:

public static async Task Run([BlobTrigger("signinsheets-pending/{name}")] Stream input, string name, [Blob("signinsheets/{name}", FileAccess.Write)] Stream output, TraceWriter log)

1. Within the **Run** method, add a new line of code at line 23 to get the **Event Key** by stripping the file extension from the name of the blob:

string eventKey = Path.GetFileNameWithoutExtension(name);

1. Add a new line of code at line 24 to create a using block using the **Stream** result from an invocation of the **ProcessStorageMessage** method:

using (MemoryStream stream = await ProcessStorageMessage(eventKey)){}

1. Within the *using* block, add a new line of code to create a new variable of type **byte[]** by invoking the **ToArray** method of the stream:

byte[] byteArray = stream.ToArray();

1. Still within the *using* block, add another line of code to invoke the **WriteAsync** method of the **output** variable passing in various metadata related to the **byteArray** variable:

await output.WriteAsync(byteArray, 0, byteArray.Length);

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

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

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Data** project and double-click the **BlobContext.cs** file.
2. In the code editor tab for the **BlobContext.cs** file, locate the **BlobContext** class:

public class BlobContext

1. Within the **BlobContext** class, locate the **UploadBlobAsync** method and delete any existing code within the method:

public async Task<ICloudBlob> UploadBlobAsync(string blobName, Stream stream)

{

}

1. Within the **UploadBlobAsync** method, add a new line of code to create a new **CloudStorageAccount** class instance:

CloudStorageAccount account = CloudStorageAccount.Parse(StorageSettings.ConnectionString);

1. Add a new line of code to create a new instance of the **CloudBlobClient** class using the **CreateCloudBlobClient** method of the **CloudStorageAccount** class:

CloudBlobClient blobClient = account.CreateCloudBlobClient();

1. Add a new line of code to get a reference to a new or existing container using the **GetContainerReference** method of the **CloudBlobClient** class:

CloudBlobContainer container = blobClient.GetContainerReference($"{StorageSettings.ContainerName}-pending");

1. Add a new line of code to invoke the **CreateIfNotExistsAsync** method of the **CloudBlobContainer** class that will create the container if it does not already exist:

await container.CreateIfNotExistsAsync();

1. Add a new line of code to get a reference to a new of existing blob using the specified blob name:
2. ICloudBlob blob = container.GetBlockBlobReference(blobName);
3. Add a new line of code to take your **stream** parameter and "rewind" the stream back to it's origin:

stream.Seek(0, SeekOrigin.Begin);

1. Add a new line of code to upload the **stream** parameter's content to the referenced blob:

await blob.UploadFromStreamAsync(stream);

1. Add a new line of code to return the updated blob as the result of the method:

return blob;

#### Task 3: Configure start up projects

1. In the **Solution Explorer** pane, right-click the **Contoso.Events** solution, and then click **Properties**.
2. Navigate to the **Startup Project** section located under the **Common Properties** header.
3. In the **Startup Project** section, locate and select the **Multiple startup projects** option.
4. Within the **Multiple startup projects** table, perform the following actions:

Locate the **Contoso.Events.Management** entry and change it's *Action* from **None** to **Start**.

Locate the **Contoso.Events.Worker** entry and change it's *Action* from **None** to **Start**.

1. Click the **OK** button to close the *Property* dialog.
2. On the **Debug** menu, click **Start Debugging**.
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 any of the events in the list. Remember the name of the event as you will click the same button again in this lab.
5. 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, it will create a new copy of the document in the **signinsheets** container.
6. **Note**: This can take up to a minute to occur.
7. Refresh the browser window that is displaying the website.
8. You should now see a section titled **Sign-In Document Already Exists**. This indicates that the sign-in document was generated successfully.
9. Do not attempt to download the document yet as that functionality has not been implemented yet.
10. Close the browser window that is displaying the website.
11. Close the console window for the Azure Functions instance.

#### Task 4: Sign in to the Azure Portal

1. On the Start screen, click the **Internet Explorer** tile.
2. Go to *(*[*https://portal.azure.com*](https://portal.azure.com/)*)*.
3. In the email address box, type the email address of your Microsoft account.
4. Click **Next**.
5. In the password box, type the password for your Microsoft account.
6. Click **Sign In**.
7. **Note**: If this is your first time logging in to the Portal, you may be prompted with a “Welcome” dialog. You can safely close this dialog and continue.

#### Task 5: Validate Azure Storage Data

1. In the **All services** blade that displays, click **Storage accounts**.
2. In the **Storage accounts** blade that displays, select the Azure Cosmos DB account instance that has a prefix of **star20532**.
3. In the *Storage account* blade, click the **Blobs** link.
4. In the **Blob service** blade, click the **signinsheets** link to view the container.
5. In the **Blob container** blade, observe the documents in the container.

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

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

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Data** project and double-click the **BlobContext.cs** file.
2. In the code editor tab for the **BlobContext.cs** file, locate the **BlobContext** class:

public class BlobContext

1. Within the **BlobContext** class, locate the **GetStreamAsync** method and delete any existing code within the method:

public async Task<DownloadPayload> GetStreamAsync(string blobId)

{

}

1. Within the **GetStreamAsync** method, add a new line of code to create a new **CloudStorageAccount** class instance:

CloudStorageAccount account = CloudStorageAccount.Parse(StorageSettings.ConnectionString);

1. Add a new line of code to create a new instance of the **CloudBlobClient** class using the **CreateCloudBlobClient** method of the **CloudStorageAccount** class:

CloudBlobClient blobClient = account.CreateCloudBlobClient();

1. Add a new line of code to get a reference to a new or existing container using the **GetContainerReference** method of the **CloudBlobClient** class:

CloudBlobContainer container = blobClient.GetContainerReference(StorageSettings.ContainerName);

1. Add a new line of code to invoke the **CreateIfNotExistsAsync** method of the **CloudBlobContainer** class that will create the container if it does not already exist:

await container.CreateIfNotExistsAsync();

1. Add a new line of code to get a reference to a new of existing blob using the specified blob name:

ICloudBlob blob = container.GetBlockBlobReference(blobId);

1. Add a new line of code to take create a new **Stream** class instance with the data from your referenced blob:

Stream blobStream = await blob.OpenReadAsync(null, null, null);

1. Add a new line of code to return the **blobStream** variable with the streamed data from your blob as the result of the method:

return new DownloadPayload { Stream = blobStream, ContentType = blob.Properties.ContentType };

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

1. In the **Solution Explorer** pane, right-click the **Contoso.Events.Management** project, and then click **Set as Startup Project**.
2. On the **Debug** menu, click **Start Debugging**.
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. On the sign-in sheet page, you should see a section titled **Sign-In Document Already Exists**.
6. Click the **Download Sign-In Sheet Stream** link to download the document as a stream from the web server.
7. Close the browser window that is displaying the website.

## Exercise 4: Specifying Permissions for the Container

#### Task 1: Securing Blob Container

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Data** project and double-click the **BlobContext.cs** file.
2. In the code editor tab for the **BlobContext.cs** file, locate the **BlobContext** class:

public class BlobContext

1. Within the **BlobContext** class, locate the **GetSecureUrlAsync** method and delete any existing code within the method:

public async Task<string> GetSecureUrlAsync(string blobId)

{

}

1. Within the **GetSecureUrlAsync** method, add a new line of code to create a new **CloudStorageAccount** class instance:

CloudStorageAccount account = CloudStorageAccount.Parse(StorageSettings.ConnectionString);

1. Add a new line of code to create a new instance of the **CloudBlobClient** class using the **CreateCloudBlobClient** method of the **CloudStorageAccount** class:

CloudBlobClient blobClient = account.CreateCloudBlobClient();

1. Add a new line of code to get a reference to a new or existing container using the **GetContainerReference** method of the **CloudBlobClient** class:

CloudBlobContainer container = blobClient.GetContainerReference(StorageSettings.ContainerName);

1. Add a new line of code to invoke the **CreateIfNotExistsAsync** method of the **CloudBlobContainer** class that will create the container if it does not already exist:

await container.CreateIfNotExistsAsync();

1. Add a new block of code to create a new instance of the **SharedAccessBlobPolicy** class setting various parameter of the policy:

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

1. Add a new line of code to create a new instance of the **BlobContainerPermissions** class setting various parameters:

BlobContainerPermissions blobPermissions = new BlobContainerPermissions{PublicAccess = BlobContainerPublicAccessType.Off};

1. Add a new line of code to add the **blobPolicy** variable with a key of **ReadBlobPolicy** to the **SharedAccessPolicies** collection of the **blobPermissions** variable:

blobPermissions.SharedAccessPolicies.Add("ReadBlobPolicy", blobPolicy);

1. Add a new line of code to update the container by invoking the **SetPermissionsAsync** method of the **container** variable:

await container.SetPermissionsAsync(blobPermissions);

#### Task 2: Implement Blob Access using Secure URL

1. Add a new line of code to generate a new SAS Token using the **ReadBlobPolicy** key and store the string result in the **sasToken** variable:

string sasToken = container.GetSharedAccessSignature(new SharedAccessBlobPolicy(), "ReadBlobPolicy");

1. Add a new line of code to get a reference to a new of existing blob using the specified blob name:

ICloudBlob blob = container.GetBlockBlobReference(blobId);

1. Add a new line of code to get the **Uri** of the **blob** variable:

Uri blobUrl = blob.Uri;

1. Add a new line of code to return the concatenation of the **sasToken** variable and the **AbsoluteUri** property of the **blobUrl** variable:

return blobUrl.AbsoluteUri + sasToken;

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

1. In the **Solution Explorer** pane, right-click the **Contoso.Events.Management** project, and then click **Set as Startup Project**.
2. On the **Debug** menu, click **Start Debugging**.
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. On the sign-in sheet page, you should see a section titled **Sign-In Document Already Exists**.
6. Click the **Use Sign-In Sheet Hyperlink** link to download the document as a stream from the web server.
7. Close the browser window that is displaying the website.

## 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 **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

Close the currently running web browser application.

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.