# Module 6: Storing Unstructured Data in Azure

# Lab: Storing Event Registration Data in Azure Cosmos DB

## Exercise 1: Populating the Sign-In Form with Registrant Names

#### 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**.

* **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 an Azure Storage Instance

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:
   1. In the **Name** field, provide the value **star20532[your name in lowercase here]**.
   2. In the **Deployment model** section, ensure that the *Resource manager* option is selected.
   3. In the **Account kind** list, ensure that the *StorageV2 (general purpose v2)* option is selected.
   4. In the **Location** list, select the region closest to your current location.
   5. Click on the **Replication** list and select the **Locally-redundant storage (LRS)** option.
   6. In the **Performance** section, ensure that the *Standard* option is selected.
   7. In the **Access tier** section, ensure that the *Hot* option is selected.
   8. In the **Secure transfer required** section, ensure that the *Enabled* option is selected.
   9. In the **Resource group** section, select the **Create new** option.
   10. In the **Resource group** section, locate the dialog box and enter the value **MOD06NSQL**.
   11. In the **Virtual networks** section, ensure that the *Disabled* option is selected.
   12. 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.

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, select the Storage account instance that has a prefix of **star20532**.
4. In the *Storage account* blade, locate the **Settings** section on the left-side of the blade and click the **Access keys** link.
5. In the **Access keys** blade, select any one of the keys and record the value in the **Connection string** field. You will use this value later in this lab.

#### Task 3: Create an Azure SQL Database Instance

1. In the navigation pane on the left side, click **All services**.
2. In the **All services** blade that displays, click **SQL databases**.
3. In the **SQL databases** blade that displays, view your list of SQL databases.
4. At the top of the **SQL databases** blade, click the **Add** button.
5. In the *SQL database* blade that displays, locate the **Database name** field and provide the value **db20532**.
6. Locate the **Resource group** section, select the **Use existing** option and then select the option **MOD06NSQL** from the list.
7. In the **Select source** list, select the value **Blank database**.
8. Click the **Server** option.
9. In the **Server** blade that displays, perform the following actions:
   1. Click **Create a new server**.
   2. In the **New Server** blade that displays, locate the **Server Name** field.
   3. In the **Server name** field, type **sv20532[*Your Name Here*]**.
   4. In the **Server admin login** field, type **testuser** as login.
   5. In the **Password** field, type **TestPa$$w0rd** as password.
   6. In the **Confirm password** field, type **TestPa$$w0rd**.
   7. In the **Location** list, select the region that is closest to your location.
   8. Click the **Select** button.
10. Back in the **SQL database** blade, click the **Pricing Tier** option.
11. In the **Resource Configuration & Pricing** blade that displays, select the **Basic** option.
12. Click **Apply** to close the blade.
13. 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.

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 **SQL Database**.
3. In the **SQL Database** blade that displays, select the SQL database instance named **db20532**.
4. In the *SQL database* blade, locate the **Settings** section on the left-side of the blade and click the **Connection strings** link.
5. 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. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **SQL servers**.
3. In the **SQL servers** blade that displays, select the SQL database instance that has a prefix of **sv20532**.
4. In the *SQL server* blade, locate the **Security** section on the left-side of the blade and click the **Firewalls and virtual networks** link.
5. 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.
6. 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 an Azure Cosmos DB Instance

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:
   1. In the **ID** field, provide the value **nosql20532[your name in lowercase here]**.
   2. In the **API** list, select the **SQL** option.
   3. In the **Resource group** section, select the **Use existing** option.
   4. In the **Resource group** section, select the value **MOD06NSQL** from the list.
   5. In the **Location** list, select the region closest to your current location.
   6. Ensure that the *Enable geo-redundancy* checkbox is un-selected.
   7. In the **Virtual networks** section, ensure that the *Disabled* option is selected.
   8. 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.

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, select the Azure Cosmos DB account instance that has a prefix of **nosql20532**.
4. In the *Azure Cosmos DB account* blade, locate the **Settings** section on the left-side of the blade and click the **Keys** link.
5. 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: Open the ASP.NET web application

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):\Mod06\Labfiles\Starter**, and then double-click **Contoso.Events.sln**.
4. On the **Build** menu, click **Build Solution**.

#### Task 6: Update Various Application Settings

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Web** project.
2. In the **Solution Explorer** pane, double-click **appsettings.json**.
3. In the JSON object, in line 13, 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 9, locate the **CosmosSettings.EndpointUrl** path. Observe that the current value is empty:

* "CosmosSettings": {  
   "DatabaseId": "EventsDatabase.Mod06",  
   "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 10, locate the **CosmosSettings.AuthorizationKey** path. Observe that the current value is empty:

* "CosmosSettings": {  
   "DatabaseId": "EventsDatabase.Mod06",  
   "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. 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.

#### Task 7: Retrieve Registrant Names in the Azure Functions Project

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

1. Within the **ProcessDocuments** class, add a new line of code at line 19 to create a new static instance of the **RegistrationContext** class:

* private static RegistrationContext \_registrationsContext = \_connection.GetCosmosContext();

1. Locate the **ProcessHttpRequestMessage** method:

* private static async Task<List<string>> ProcessHttpRequestMessage(string eventKey)

1. Within the **ProcessHttpRequestMessage** method, add a new line of code at line 40 configure the connection to Azure Cosmos DB using the **ConfigureConnectionAsync** method of the **RegistrationContext** class:

* await \_registrationsContext.ConfigureConnectionAsync();

1. Locate the line of code at line 44 that stores an empty collection of string values in the **registrants** variable:

* List<string> registrants = new List<string>();

1. Replace the line of code at line 44 with a new line of code that invokes the **GetRegistrantsForEvent** method of the **RegistrationContext** class:

* List<string> registrants = await \_registrationsContext.GetRegistrantsForEvent(eventKey);

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

## Exercise 2: Updating the Events Website to use Azure Cosmos DB

#### Task 1: Implement RegistrationContext 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 **RegistrationContext.cs** file.
2. In the code editor tab for the **RegistrationContext.cs** file, locate the **RegistrationContext** class:

* public class RegistrationContext

1. Within the **RegistrationContext** class, add a new line of code to create a new property of type **Database**:

* protected Database Database { get; set; }

1. Within the **RegistrationContext** class, add a new line of code to create a new property of type **DocumentCollection**:

* protected DocumentCollection Collection { get; set; }

1. Within the **RegistrationContext** class, add a new line of code to create a new property of type **DocumentClient**:

* protected DocumentClient Client { get; set; }

1. Within the **RegistrationContext** class, locate the existing **RegistrationContext** constructor:

* public RegistrationContext(IOptions<CosmosSettings> cosmosSettings)  
  {  
   CosmosSettings = cosmosSettings.Value;  
  }

1. Within the constructor, add a new line of code to create a new **DocumentClient** instance and save it to the *Client* property:

* Client = new DocumentClient(new Uri(CosmosSettings.EndpointUrl), CosmosSettings.AuthorizationKey);

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

* public async Task ConfigureConnectionAsync()  
  {  
    
  }

1. Within the **ConfigureConnectionAsync** method, add a new line of code to create a new **Database** instance and save it to the **Database** property:

* Database = await Client.CreateDatabaseIfNotExistsAsync(new Database { Id = CosmosSettings.DatabaseId });

1. Add a new line of code to create a new **DocumentCollection** instance and save it to the **Collection** property:

* Collection = await Client.CreateDocumentCollectionIfNotExistsAsync(Database.SelfLink, new DocumentCollection { Id = CosmosSettings.CollectionId });

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

* public async Task<int> GetRegistrantCountAsync()  
  {  
    
  }

1. Within the **GetRegistrantCountAsync** method, add a new line of code to create a new instance of the **FeedOptions** class with it's **EnableCrossPartitionQuery** property set to a value of true:

* FeedOptions options = new FeedOptions { EnableCrossPartitionQuery = true };

1. Add a new line of code to create a query that gets the count of registrants and returns a collection of integer values:

* IDocumentQuery<int> query = Client.CreateDocumentQuery<int>(Collection.SelfLink, "SELECT VALUE COUNT(1) FROM registrants", options).AsDocumentQuery();

1. Add a new block of code to create an integer variable named **count**, a while loop that iterates over the **HasMoreResults** property of the **IDocumentQuery<>** class and returns the final value of the **count** variable:

* int count = 0;  
  while (query.HasMoreResults)  
  {  
    
  }  
  return count;

1. Within the while loop, add a line of code to invoke the **ExecuteNextAsync<>** method of the **IDocumentQuery<>** class and store it's result in a variable named **results** of type **FeedResponse<>**:

* FeedResponse<int> results = await query.ExecuteNextAsync<int>();

1. Still within the while loop, add a line of code to increment the **count** variable by the result of invoking the **Sum** LINQ method on the collection stored in the **results** variable:

* count += results.Sum();

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

* public async Task<List<string>> GetRegistrantsForEvent(string eventKey)  
  {  
    
  }

1. Within the **GetRegistrantsForEvent** method, add a line of code that uses LINQ to get registrants for a specific event using the value of the **eventKey** parameter and de-serialize those registrants using the **GeneralRegistration** class:

* IDocumentQuery<GeneralRegistration> query = Client.CreateDocumentQuery<GeneralRegistration>(Collection.SelfLink).Where(r => r.EventKey == eventKey).AsDocumentQuery();

1. Add a new block of code to create a **List<>** variable named **registrants**, a while loop that iterates over the **HasMoreResults** property of the **IDocumentQuery<>** class and returns the final value of the **registrants** variable:

* List<string> registrants = new List<string>();  
  while (query.HasMoreResults)  
  {  
    
  }  
  return registrants;

1. Within the while loop, add a line of code to invoke the **ExecuteNextAsync<>** method of the **IDocumentQuery<>** class and store it's result in a variable named **results** of type **FeedResponse<>**:

* FeedResponse<GeneralRegistration> results = await query.ExecuteNextAsync<GeneralRegistration>();

1. Still within the while loop, add a line of code that invokes the **Select** LINQ method to project two properties of the **GeneralRegistration** class as a single string and store the resulting collection in a variable named **resultNames** of type **IEnumerable<>**:

* IEnumerable<string> resultNames = results.Select(r => $"{r.FirstName} {r.LastName}");

1. Still within the while loop, add another line of code to append the **registrants** collection with the values in the **resultNames** collection using the **AddRange** method of the **List<>** class.

* registrants.AddRange(resultNames);

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

* public async Task<string> UploadEventRegistrationAsync(dynamic registration)  
  {  
    
  }

1. Within the **UploadEventRegistrationAsync** method, add a line of code to invoke the **CreateDocumentAsync** method of the **Client** property of type **DocumentClient** and save the result as a variable named **response** of type **ResourceResponse<Document>**. This method will upload the document to Azure Cosmos DB:

* ResourceResponse<Document> response = await Client.CreateDocumentAsync(Collection.SelfLink, registration);

1. Add another line of code to use dot notation to access the **Resource** property (of type **Document**) of the **response** variable and the **Id** property of the **Document** instance. Return the string **Id** value as the result of the current method:

* return response.Resource.Id;

## Exercise 3: Verify that the Events Web Site is using Azure Cosmos DB for Registrations

#### Task 1: Run the ASP.NET Web Application to View Events from the Local SQL Database

1. In the **Solution Explorer** pane, right-click the **Contoso.Events.Web** 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 any of the events in the list.
5. Record the value of the unique **Event Key** of the event you selected. You will use this value later in the lab.

* **Note**: You can see this unique event key by looking at the URL for the event. For example, if the URL is http://localhost:52127/events/detail/fy18septembergeneralconference09, the unique event key would be fy18septembergeneralconference09.

1. On the event web page, click the **Register Now** button.
2. Fill out all of the fields in the registration form and click Submit.
3. Record the value of the **Registration Id** displayed after submitting an event registration. You will use this value later in the lab.
4. Close the browser window that is displaying the website.
5. In the **Solution Explorer** pane, right-click the **Contoso.Events.Worker** project, and then click **Set as Startup Project**.
6. On the **Debug** menu, click **Start Debugging**.
7. In the command prompt, you will see a single **Function URL** displayed in *green text* for your **ProcessDocuments** Function endpoint. Record this value as you will use this value later in this lab.
8. On the Start screen, click the **Internet Explorer** tile.
9. In the browser address bar, paste in the **Function URL** you copied earlier in this lab. Use the **Enter** key to issue a **HTTP GET** request to the URL.
10. Return to the console application. You will observe various logging error messages. Most importantly, you will see see a message indicating that there were no registrants for the event key you specified (**null**):

* [00/00/0000 0:00:00 PM] Registrants:

1. Return to your open browser window and update your URL by adding a new *query string* parameter named **eventKey** and using the unique **Event Key** you copied earlier in this lab. Use the **Enter** key to issue a **HTTP GET** request to the URL.

* **Note**: For example, if your *Endpoint URL* is http://localhost:7071/api/ProcessDocuments and your **Event Key** is **fy18septembergeneralconference09**, your new URL would be http://localhost:7071/api/ProcessDocuments?eventKey=fy18septembergeneralconference09.

1. Return to the console application again. Observe that the registration you created earlier is included in the list of registrations. Below is an example list of event registrations:

* [00/00/0000 0:00:00 PM] Registrants: Example Person, Demo User, Unique Individual

1. Close the browser window that you are using to send requests.
2. Close the console window for the Azure Functions instance.

#### Task 2: 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. In the password box, type the password for your Microsoft account.
5. Click **Sign In**.

* **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: Validate Azure Cosmos DB Data

1. In the **All services** blade that displays, click **Azure Cosmos DB**.
2. In the **Azure Cosmos DB** blade that displays, select the Azure Cosmos DB account instance that has a prefix of **nosql20532**.
3. In the *Azure Cosmos DB account* blade, locate the **Data Explorer** link on the left-side of the blade.
4. In the **Data Explorer** pane, expand the **EventsDatabase.Mod06** database node and then expand the **RegistrationCollection** collection node.
5. Click the **New SQL Query** button at the top of the **Data Explorer** pane.
6. In the query tab, replace the contents of the *query editor* with the following SQL query:

* SELECT \* FROM registrations ORDER BY registrations.\_ts DESC
* **Note**: This query will match all documents but only return a page of results at at time. We are ordering by the timestamp field (\*\*\_ts\*\*) so that you can see the registrations you created first.

1. Click the **Execute Query** button in the query tab to run the query.
2. In the **Results** pane, observe the results of your query.

* **Note**: You should now see various JSON documents representing each event registration. There are registrations you manually created and registrations created automatically by the application when it first launched.

1. In the query tab, replace the contents of the *query editor* with the following SQL query replacing the {registration\_id} placeholder with the value of the **Registration Id** you recorded earlier in this lab:

* SELECT VALUE { first: r.FirstName, last: r.LastName } FROM registrations r WHERE r.id = '{registration\_id}'

1. Click the **Execute Query** button in the query tab to run the query.
2. In the **Results** pane, observe the results of your query.
3. In the query tab, replace the contents of the *query editor* with the following SQL query replacing the {event\_key} placeholder with the value of the **Event Key** you recorded earlier in this lab:

* SELECT VALUE CONCAT(r.FirstName, ' ', r.LastName) FROM registrations r WHERE r.EventKey = '{event\_key}'

1. Click the **Execute Query** button in the query tab to run the query.
2. In the **Results** pane, observe the results of your query.

## 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 **MOD06NSQL** *Resource Group*:

* az group delete --name MOD06NSQL --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.

©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.