Module 04 (CSR): Develop solutions that use Cosmos DB storage

1 Hr 17 Min Remaining

Instructions Resources Help  100%

Required Lab Setup

Sign in to the lab virtual machine

1. Hello Altaf Hussain, log on to [AZ-204T00A-SEA-DEV](https://labclient.labondemand.com/Instructions/096e3a82-07ce-4bf3-91a3-ad375d3ad595?rc=10) click [Ctrl+Alt+Delete](https://labclient.labondemand.com/Instructions/096e3a82-07ce-4bf3-91a3-ad375d3ad595?rc=10) to activate the **Ctrl + Alt + Delete** sequence and bring up the logon page.
2. Sign in as Admin with the password Pa55w.rd

Any links like the one above will send Ctrl+Alt+Delete to the selected machine. This can also be done the **Commands** menu (lightning bolt) in the upper-left hand corner of the screen.

The **Lightning Bolt** in the upper left hand of the screen can also be used to **Copy /Paste** strings and sentences from the Instructions **into the VM** rather than typing them out.

* + To accomplish this simply Highligh\*\*t the sentence in the instructions you wish to type text. **CTRL + C** (Copy)
  + Click the lightning bolt icon, in the Dropdown menu select **Type Text** > **Type Clipboard Text**
  + Click into the Type Text window, **CTRL + V** (Paste) to Paste the sentence into the window
  + Select OK

This Lab supports **Redirect Clipboard** functionality in addition to TypeText.

You can use **Redirect Clipboard** to quickly input code blocks and other strings from the Instructions and elsewhere into the virtual machine using standard copy and paste from your local machine's Clipboard directly into the VM. You may need to click **Allow** in your browser to allow access to your local Clipboard.

Note that due to an issue within **Azure Cloudshell**, use your mouse: **Right-click + Paste** instead of **CTRL+V** when using **Cloudshell** inside the VM.

Prepare cloud shell for later use

Mount storage in Azure

1. Sign in to the Azure Portal https://portal.azure.com using the below credentials:

|  |  |
| --- | --- |
| Username | LabUser-23788072@cloudslice.onmicrosoft.com |
| Password | a#BqhJN21! |

1. In the toolbar at the top of the Azure portal, select the **Cloud Shell** icon.
2. In the Welcome to Azure Cloud Shell dialog, select **Bash**.
3. On the you have no storage mounted screen select **Show advanced settings**.
4. In the advanced settings screen, fill in the following fields, then click **Create Storage**:

|  |  |
| --- | --- |
| Resource Group | Use existing (**PolygotData-lod23788072**) |
| Cloud Shell Region | **East US** |
| Storage account (Create new) | cloudshell23788072 |
| File share (create new) | shellstorage |

1. After the cloud shell initializes and puts you at a text prompt, exit the shell.

At the end of this lab, you can skip the **Clean Up** exercise directing you to remove the resources from your Subscription or Resource Group(s). The clean up is handled automatically, after ending your lab.

Download the lab files

Download Files

This will copy the latest lab files from GitHub and configure the directories as needed for the remainder of the lab exercises. You will see a 'Success!' message above, once the process is complete.

Lab Credentials:

* Username: LabUser-23788072@cloudslice.onmicrosoft.com
* Password: a#BqhJN21!

Sites used:

* https://portal.azure.com/#home
* https://shell.azure.com

All the resource groups you require in this lab have been created for you as part of the lab setup. You can safely ignore any steps that ask you to create a resource group. Please use or select a pre-existing resource group that has a similar name to the one you are asked to create. If no similarly named resource group exists, use any existing resource group.

For access to Azure, use the following credentials:

* Username: LabUser-23788072@cloudslice.onmicrosoft.com
* Password: a#BqhJN21!

You can see your available Resource Groups by checking the Resources tab above.

Lab 04: Develop solutions that use Azure Cosmos DB

Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes that occur after the development of this training content. As a result, the lab instructions and lab steps might not align correctly.

Microsoft updates this training course when the community alerts us to needed changes. However, cloud updates occur frequently, so you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

Instructions

Before you start

Sign in to the lab environment

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

* Username: **Admin**
* Password: **Pa55w.rd**

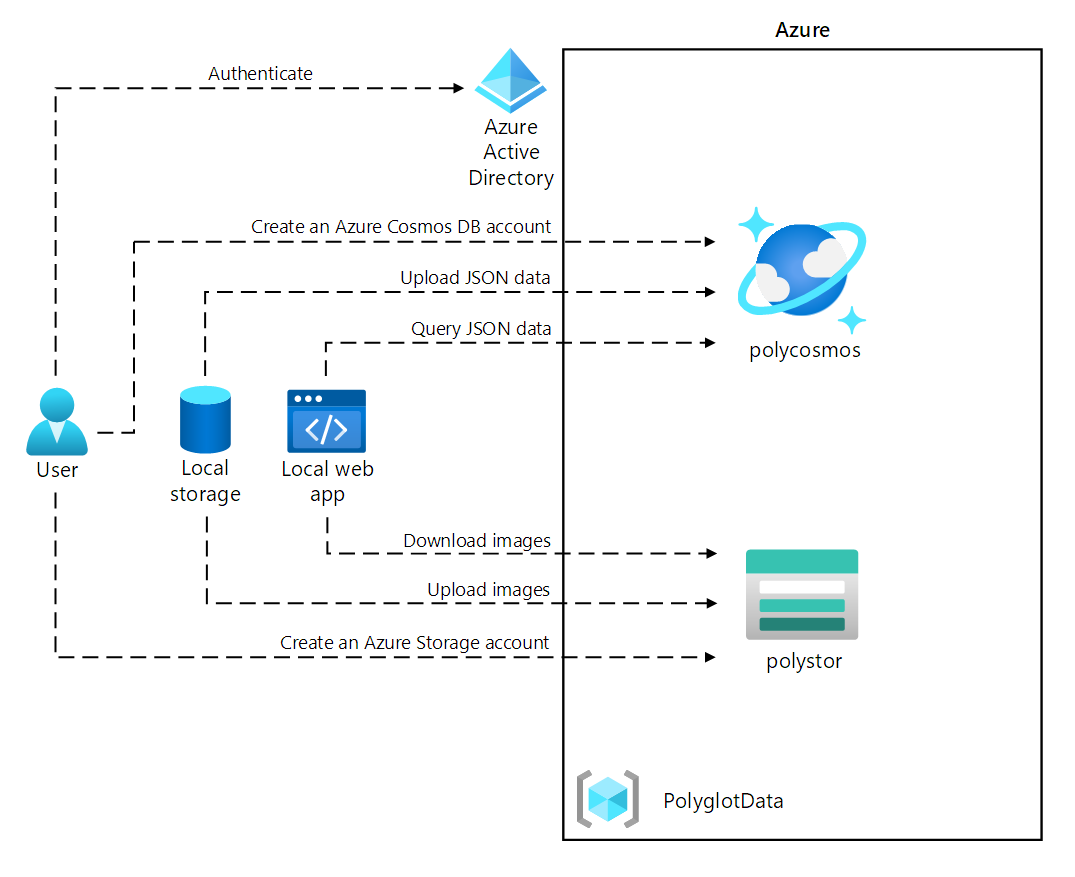
**Note**: Your instructor will provide instructions to connect to the virtual lab environment.

Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab, including:

* Microsoft Edge
* File Explorer
* Visual Studio Code

Architecture diagram



Exercise 1: Creating data store resources in Azure

Task 1: Open the Azure portal

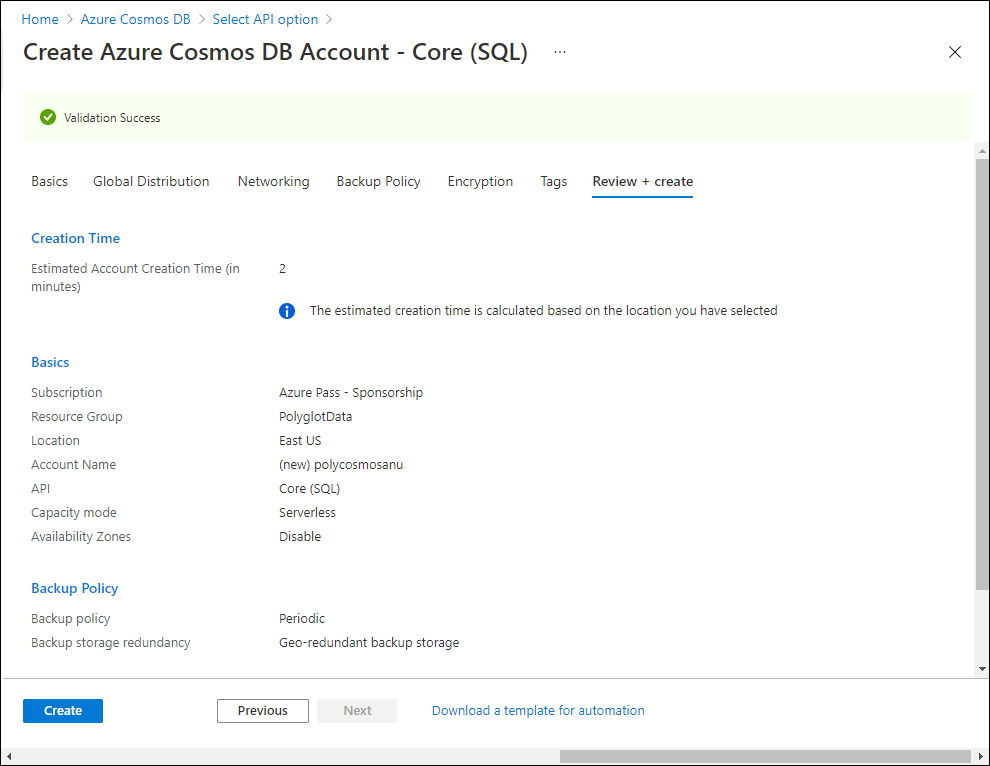
1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, browse to the Azure portal https://portal.azure.com, and then sign in with the account you will be using for this lab.

**Note**: If this is your first time signing in to the Azure portal, you'll be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

Task 2: Create an Azure Cosmos DB account resource

1. In the Azure portal, use the **Search resources, services, and docs** text box to search for **Azure Cosmos DB** and then in the list of results, select **Azure Cosmos DB**.
2. On the **Azure Cosmos DB** blade, select **+ Create**.
3. On the **Select API option** blade, select **Create** in the **Core (SQL) - Recommended** box.
4. On the **Basics** tab of the **Create Azure Cosmos DB Account - Core (SQL)** blade, perform the following actions, and then select **Review + Create**:

| **Setting** | **Action** |
| --- | --- |
| **Subscription** list | Retain defaults. |
| **Resource group** section | Use existing |
| **Name** text box | Use **PolygotData-lod23788072** |
| **AccountName** text box | Enter **polycosmos***[yourname]*. |
| **Location** drop-down list | Select an Azure region that is closest to the location of your lab computer and where you can create a Cosmos DB account. |
| **Capacity mode** section | Select **Serverless**. |

1. The following screenshot displays the configured settings on the **Create Azure Cosmos DB Account - Core (SQL)** blade.
2. 
3. On the **Review + Create** tab of the **Create Azure Cosmos DB Account - Core (SQL)** blade, review the options that you selected during the previous steps.
4. Select **Create** to create the Azure Cosmos DB account by using your specified configuration.

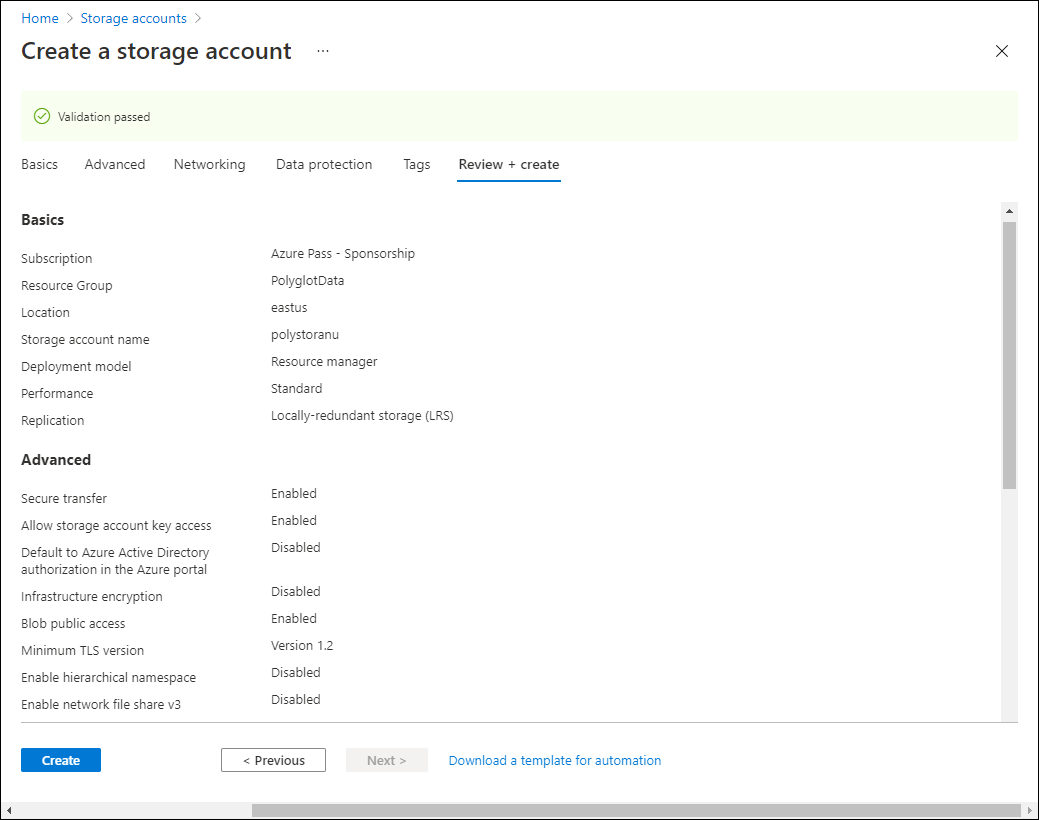
**Note**: Wait for the creation task to complete before you move forward with this lab.

1. Select **Go to resource**.
2. On the **Azure Cosmos DB account** blade, find the **Settings** section, and then select the **Keys** link.
3. In the **Keys** pane, on the **Read-write Keys** tab, record the values of the **URI**, **PRIMARY KEY**, and **PRIMARY CONNECTION STRING** text boxes. You'll use these values later in this lab.

Task 3: Create an Azure Storage account resource

1. In the Azure portal, use the **Search resources, services, and docs** text box to search for **Storage accounts** and, in the list of results, select **Storage accounts**.
2. On the **Storage accounts** blade, select **+ Create**.
3. On the **Basics** tab of the **Create a storage account** blade, perform the following actions, and then select **Review + Create**:

| **Setting** | **Action** |
| --- | --- |
| **Subscription** list | Retain defaults |
| **Resource group** section | Select **PolygotData-lod23788072**. |
| **Storage account name** text box | Enter **polystor***[yourname]*. |
| **Region** drop-down list | Select the same region where you created the Cosmos DB account earlier in this exercise. |
| **Performance** section | Select **Standard**. |
| **Redundancy** drop-down list | Select **Locally-redundant storage (LRS)**. |

1. The following screenshot displays the configured settings on the **Create a storage account** blade.
2. 
3. On the **Review + Create** tab of the **Create a storage account** blade, review the options that you selected during the previous steps.
4. Select **Create** to create the storage account by using your specified configuration.

**Note**: Wait for the creation task to complete before you proceed with this lab.

Review

In this exercise, you created the Azure resources that you'll need for the polyglot data solution you'll implement in this lab. The Azure resources you created include an Azure Cosmos DB account and an Azure Storage account.

Exercise 2: Review and upload data

Task 1: Upload images to Azure Blob Storage

1. In the Azure portal's navigation pane, navigate back to the **Storage accounts** blade, and then select the **polystor***[yourname]* storage account that you created in this lab's previous exercise.
2. On the **polystor***[yourname]* storage account blade, select the **Containers** link in the **Data storage** section.
3. In the **Containers** section, select **+ Container**.
4. In the **New container** pop-up window, perform the following actions, and then select **Create**:

| **Setting** | **Action** |
| --- | --- |
| **Name** text box | Enter **images**. |
| **Public access level** drop-down list | Select **Blob (anonymous read access for blobs only)**. |

1. Back in the **Containers** section, select the newly created **images** container.
2. Find the **Settings** section on the **Container** blade, and then select the **Properties** link.
3. In the **Properties** pane, note and record the value in the **URL** text box. You'll use this value later in this lab.
4. Find and select the **Overview** link on the blade.
5. On the blade, select **Upload**.
6. In the **Upload blob** pop-up, perform the following actions:
   1. In the **Files** section, select the **Folder** icon.
   2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\04\Starter\Images**, select all 42 individual **.jpg** image files, and then select **Open**.
   3. Ensure that **Overwrite if files already exist** is selected, and then select **Upload**.

**Note**: Wait for all blobs to upload before you continue with this lab.

Task 2: Review JSON data

1. From the lab computer, start Visual Studio Code.
2. From the **File** menu, select **Open File**, browse to **Allfiles (F):\Allfiles\Labs\04\Starter\AdventureWorks\AdventureWorks.Upload**, select **models.json**, and then select **Open**.
3. Review the format of the **models.json** file and note that it contains an array of JSON objects, with a nested array of objects that are part of the **Products** property.

**Note**: This will determine the classes you'll define to deserialize the JSON file's contents before uploading it to a Cosmos DB collection.

1. Within the **models.json** file, note that one of the properties is named **Category**.

**Note**: You'll use the **Category** property to define partitioning of the target Cosmos DB collection.

1. Close Visual Studio Code.

Task 3: Create a Cosmos DB database and collection, and perform a JSON data upload

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\04\Starter\AdventureWorks**, and then select **Select Folder**.
4. In the **Visual Studio Code** window, on the Menu Bar, select **Terminal** and then select **New Terminal**.
5. In the terminal, verify that the current directory is set to **AdventureWorks** (or change it to that if it's not), and then run the following command to switch your terminal context to the **AdventureWorks.Upload** folder:

cd .\AdventureWorks.Upload\

**Note**: Before you perform the next step, open Windows Explorer and remove the **Read-only** attribute from the file **Allfiles (F):\Allfiles\Labs\04\Starter\AdventureWorks\AdventureWorks.Upload\AdventureWorks.Upload.csproj**

1. From the terminal prompt, run the following command to add the Azure Cosmos DB .NET client library to the currently opened project:

dotnet add package Microsoft.Azure.Cosmos --version 3.20.1

**Note**: The **dotnet add package** command will add the **Microsoft.Azure.Cosmos** package from **NuGet**. For more information, refer to [Microsoft.Azure.Cosmos](https://www.nuget.org/packages/Microsoft.Azure.Cosmos/3.20.1" \t "_blank).

1. Observe the results of the build printed in the terminal. The build should complete successfully with no errors or warning messages.
2. In the **Explorer** pane of the **Visual Studio Code** window, expand the **AdventureWorks.Upload** project.
3. Open the **Program.cs** file.
4. In the **Program.cs** file, review the **using** directives and note that they include **Microsoft.Azure.Cosmos**, **System.IO;**, **System.Text.Json**, **System.Threading.Tasks**, and **System.Collections.Generic**. This enables asynchronous upload of JSON items from a local file on your lab computer to a collection in a Cosmos DB database.
5. In the **Program.cs** file, on line 14, set the value of **EndpointUrl** by replacing the empty string with the **URI** property of the Cosmos DB account that you recorded earlier in this lab. Ensure that the value is enclosed in double quotes.
6. On line 15, set the value of **AuthorizationKey** by replacing the empty string with the **PRIMARY KEY** property of the Cosmos DB account that you recorded earlier in this lab. Ensure that the value is enclosed in double quotes.
7. On line 18, set the value of **PartitionKey** by replacing the empty string with **"/Category"**.
8. On line 19, set the value of **JsonFilePath** by replacing the empty string with "F:\\Allfiles\\Labs\\04\\Starter\\AdventureWorks\\AdventureWorks.Upload\\models.json".
9. Within the try block, note the invocation of the **CreateDatabaseIfNotExistsAsync** method of the **CosmosClient** class. This will create a database if one doesn't already exist.
10. Note the invocation of the **DefineContainer** method of the **Database** class. This will create a container that will host the JSON items if one doesn't already exist.

**Note**: The **DefineContainer** method includes a cost-minimizing option whereby you can modify the default indexing policy (which automatically indexes all attributes).

1. Note the **using** statement that relies on a **StreamReader** object to read JSON items from a text file and deserializes them into objects of the **Model** class defined further in the **Program.cs** file.
2. Note the foreach loop that iterates over the collection of deserialized objects and asynchronously inserts each of them into the target collection.
3. Review the **Model** and **Product** classes that reflect the format of the objects stored in the JSON-formatted file you reviewed earlier in this lab.
4. Save and close the **Program.cs** file.

**Note**: Select **Overwrite** if you received a prompt that the file is read-only.

1. In terminal, run the following command to restore any missing NuGet packages and build the project in the folder:

dotnet build

**Note**: The **dotnet build** command will automatically restore any missing NuGet packages prior to building all projects in the folder.

1. From the terminal prompt, run the following command to run the .NET Core console application:

dotnet run

**Note**: The **dotnet run** command will automatically build any changes to the project and then start the web application without a debugger attached. The command will output the messages indicating the data load's progress, including the number of items inserted into the target collection and the duration of the insert operation.

1. Observe the results of running the command printed in the terminal. The run should complete successfully, displaying the message about there being 119 items inserted into the target Cosmos DB collection.
2. Select **Kill Terminal** (the **Recycle Bin** icon) to close the terminal pane and any associated processes.

Task 4: Validate JSON data upload

1. On your lab computer, switch to the **Microsoft Edge** browser window displaying the Azure portal.
2. In the Azure portal, select the **Search resources, services, and docs** text box, in the **Recent resources** list, select the **polycosmos***[yourname]* Azure Cosmos DB account that you created earlier in this lab.
3. On the **Azure Cosmos DB account** blade, find and select the **Data Explorer** link on the blade.
4. In the **Data Explorer** pane, expand the **Retail** database node.
5. Expand the **Online** container node, and then select **New SQL Query**.

**Note**: The label for this option might be hidden. You can display labels by hovering over the icons in the **Data Explorer** pane.

1. On the query tab, enter the following text:

SELECT \* FROM models

1. Select **Execute Query**, and then observe the list of JSON items returned by the query.
2. Back in the query editor, replace the existing text with the following text:

SELECT VALUE COUNT(1) FROM models

1. Select **Execute Query**, and then observe the result of the **COUNT** aggregate operation.
2. Switch back to the **Visual Studio Code** window.

Review

In this exercise, you used the .NET SDK for Azure Cosmos DB to insert data into Azure Cosmos DB. The web application that you implement next will use this data.

Exercise 3: Configure a .NET web application

Task 1: Update references to data stores and build the web application

1. In the **Explorer** pane of the **Visual Studio Code** window, expand the **AdventureWorks.Web** project.
2. Open the **appsettings.json** file.
3. In the JSON object on line 3, find the **ConnectionStrings.AdventureWorksCosmosContext** path. Note that the current value is empty:
4. "ConnectionStrings": {
5. "AdventureWorksCosmosContext": "",

},

1. Update the value of the **AdventureWorksCosmosContext** property by setting its value to the **PRIMARY CONNECTION STRING** of the Azure Cosmos DB account that you recorded earlier in this lab.
2. In the JSON object on line 6, find the **Settings.BlobContainerUrl** path. Note that the current value is empty:
3. "Settings": {
4. "BlobContainerUrl": "",
5. ...

}

1. Update the **BlobContainerUrl** property by setting its value to the **URL** property of the Azure Storage blob container named **images** that you recorded earlier in this lab.
2. Save the **appsettings.json** file and close it.

**Note**: Select **Overwrite** if you received a prompt that the file is read-only.

1. In the **Visual Studio Code** window, select **AdventureWorks.Context**, activate the shortcut menu, and then select **Open in Integrated Terminal**.

**Note**:Before you perform the next step, open Windows Explorer and remove the Read-only attribute from the file **Allfiles (F):\Allfiles\Labs\04\Starter\AdventureWorks\AdventureWorks.Context\AdventureWorks.Context.csproj**

1. From the terminal prompt, verify that the current directory is set to **AdventureWorks.Context** (or change it to that if it's not), and then run the following command to import **Microsoft.Azure.Cosmos** from NuGet:

dotnet add package Microsoft.Azure.Cosmos --version 3.20.1

1. From the terminal prompt, run the following command to build the .NET web application:

dotnet build

1. Observe the results of the build printed in the terminal. The build should complete successfully with no errors or warning messages.

Task 2: Configure connectivity to Azure Cosmos DB

1. In the **Explorer** pane of the **Visual Studio Code** window, expand the **AdventureWorks.Context** project.
2. From the shortcut menu of the **AdventureWorks.Context** folder node, select **New File**.
3. At the new file prompt, enter **AdventureWorksCosmosContext.cs**.
4. From the code editor tab for the **AdventureWorksCosmosContext.cs** file, add the following lines of code to import the **AdventureWorks.Models** namespace from the referenced **AdventureWorks.Models** project:

using AdventureWorks.Models;

1. Add the following lines of code to import the **Microsoft.Azure.Cosmos** and **Microsoft.Azure.Cosmos.Linq** namespaces from the **Microsoft.Azure.Cosmos** package imported from NuGet:
2. using Microsoft.Azure.Cosmos;

using Microsoft.Azure.Cosmos.Linq;

1. Add the following lines of code to include **using** directives for the built-in namespaces that this file will use:
2. using System;
3. using System.Collections.Generic;
4. using System.Linq;

using System.Threading.Tasks;

1. Enter the following code to add an **AdventureWorks.Context** namespace block:
2. namespace AdventureWorks.Context
3. {

}

1. Within the **AdventureWorks.Context** namespace, enter the following code to create a new **AdventureWorksCosmosContext** class:
2. public class AdventureWorksCosmosContext
3. {

}

1. Update the declaration of the **AdventureWorksCosmosContext** class by adding a specification indicating that this class will implement the **IAdventureWorksProductContext** interface:
2. public class AdventureWorksCosmosContext : IAdventureWorksProductContext
3. {

}

1. Within the **AdventureWorksCosmosContext** class, enter the following code to create a new read-only *Container* variable named **\_container**:

private readonly Container \_container;

1. Within the **AdventureWorksCosmosContext** class, add a new constructor with the following signature:
2. public AdventureWorksCosmosContext(string connectionString, string database = "Retail", string container = "Online")
3. {

}

1. Within the constructor, add the following block of code to create a new instance of the **CosmosClient** class, and then obtain both a **Database** and **Container** instance from the client:
2. \_container = new CosmosClient(connectionString)
3. .GetDatabase(database)

.GetContainer(container);

1. Within the **AdventureWorksCosmosContext** class, add a new **FindModelAsync** method with the following signature:
2. public async Task<Model> FindModelAsync(Guid id)
3. {

}

1. Within the **FindModelAsync** method, add the following blocks of code to create a LINQ query, transform it into an iterator, iterate over the result set, and then return the single item in the result set:
2. var iterator = \_container.GetItemLinqQueryable<Model>()
3. .Where(m => m.id == id)
4. .ToFeedIterator<Model>();
5. List<Model> matches = new List<Model>();
6. while (iterator.HasMoreResults)
7. {
8. var next = await iterator.ReadNextAsync();
9. matches.AddRange(next);
10. }

return matches.SingleOrDefault();

1. Within the **AdventureWorksCosmosContext** class, add a new **GetModelsAsync** method with the following signature:
2. public async Task<List<Model>> GetModelsAsync()
3. {

}

1. Within the **GetModelsAsync** method, add the following blocks of code to run an SQL query, get the query result iterator, iterate over the result set, and then return the union of all results:
2. string query = $@"SELECT \* FROM items";
3. var iterator = \_container.GetItemQueryIterator<Model>(query);
4. List<Model> matches = new List<Model>();
5. while (iterator.HasMoreResults)
6. {
7. var next = await iterator.ReadNextAsync();
8. matches.AddRange(next);
9. }

return matches;

1. Within the **AdventureWorksCosmosContext** class, add a new **FindProductAsync** method with the following signature:
2. public async Task<Product> FindProductAsync(Guid id)
3. {

}

1. Within the **FindProductAsync** method, add the following blocks of code to run an SQL query, get the query result iterator, iterate over the result set, and then return the single item in the result set:
2. string query = $@"SELECT VALUE products
3. FROM models
4. JOIN products in models.Products
5. WHERE products.id = '{id}'";
6. var iterator = \_container.GetItemQueryIterator<Product>(query);
7. List<Product> matches = new List<Product>();
8. while (iterator.HasMoreResults)
9. {
10. var next = await iterator.ReadNextAsync();
11. matches.AddRange(next);
12. }

return matches.SingleOrDefault();

1. Save and close the **AdventureWorksCosmosContext.cs** file.
2. From the terminal prompt, with the current directory set to **AdventureWorks.Context**, run the following command to build the .NET web application:

dotnet build

**Note**: If there are any build errors, review the **AdventureWorksCosmosContext.cs** file in the **Allfiles (F):\Allfiles\Labs\04\Solution\AdventureWorks\AdventureWorks.Context** folder.

Task 3: Review the .NET application startup logic

1. In the **Explorer** pane of the **Visual Studio Code** window, expand the **AdventureWorks.Web** project.
2. Open the **Startup.cs** file.
3. In the **Startup** class, note the existing **ConfigureProductService** method:
4. public void ConfigureProductService(IServiceCollection services)
5. {
6. services.AddScoped<IAdventureWorksProductContext, AdventureWorksCosmosContext>(provider =>
7. new AdventureWorksCosmosContext(
8. \_configuration.GetConnectionString(nameof(AdventureWorksCosmosContext))
9. )
10. );

}

**Note**: The product service uses Cosmos DB as its database.

1. Close the **Startup.cs** file without making any modifications.

Task 4: Validate that the .NET application successfully connects to data stores

1. In Visual Studio Code, from the terminal prompt, run the following command to switch your terminal context to the **AdventureWorks.Web** folder:

cd ..\AdventureWorks.Web\

1. From the terminal prompt, run the following command to run the ASP.NET web application:

dotnet run

**Note**: The **dotnet run** command will automatically build any changes to the project and then start the web application without a debugger attached. The command will output the URL of the running application and any assigned ports.

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, browse to the currently running web application http://localhost:5000.
3. In the web application, observe the list of models displayed from the front page.
4. Find the **Touring-1000** model, and then select **View Details**.
5. On the **Touring-1000** product detail page, review the listing of options.
6. Close the browser window displaying your web application.
7. Switch to the **Visual Studio Code** window, and then select **Kill Terminal** (the **Recycle Bin** icon) to close the currently open terminal and any associated processes.

Review

In this exercise, you wrote C# code to query an Azure Cosmos DB collection by using the .NET SDK.

Exercise 4: Clean up your subscription

You can skip the **Clean Up** exercise directing you to remove the resources from your Subscription or Resource Group(s). The clean up is handled automatically, after ending your lab.

Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon Cloud Shell icon to open a new Bash session. If Cloud Shell defaults to a PowerShell session, select **PowerShell** and, in the drop-down menu, select **Bash**.

**Note**: If this is the first time you're starting **Cloud Shell**, when prompted to select either **Bash** or **PowerShell**, select **PowerShell**. When you are presented with the **You have no storage mounted** message, select the subscription you're using in this lab, and then select **Create storage**.

Task 2: Delete resource groups

1. In the **Cloud Shell** pane, run the following command to delete the **PolyglotData** resource group:

az group delete --name PolyglotData --no-wait --yes

**Note**: The command executes asynchronously (as determined by the *--no-wait* parameter). Therefore, while you'll be able to run another Azure CLI command immediately after, within the same Bash session, it'll take a few minutes before the resource groups are actually removed.

1. Close the **Cloud Shell** pane in the portal.

Task 3: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

Review

In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

Congratulations!

You have successfully completed this **Lab** press **End** to end your lab.

70% Tasks Complete

End

Live Chat