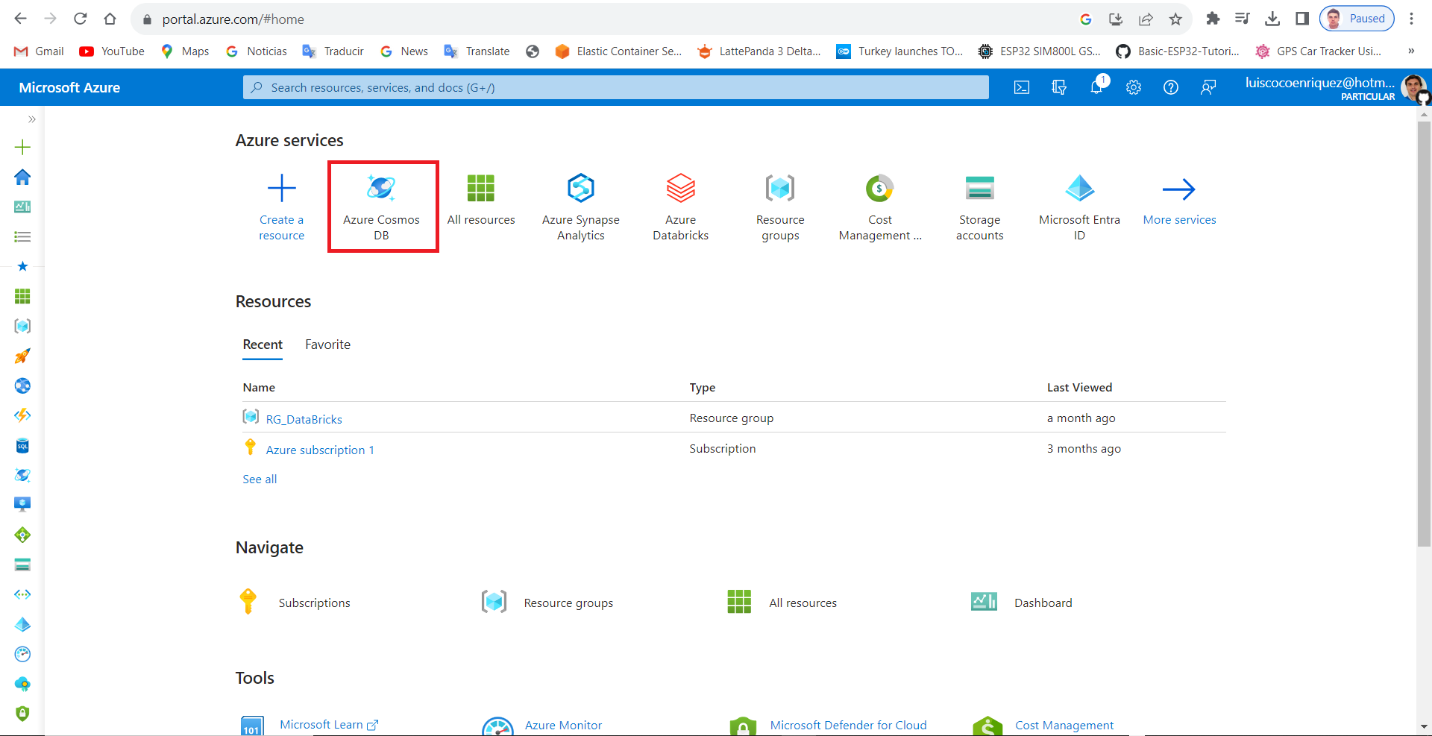
**Azure CosmosDB**

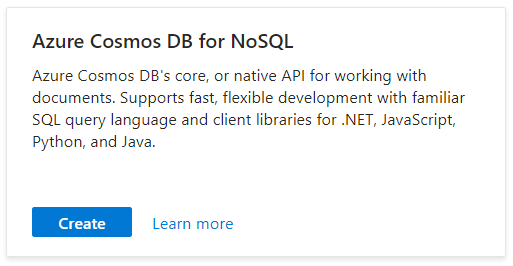


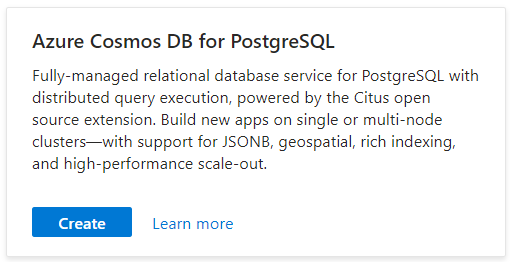
A screenshot of a computer

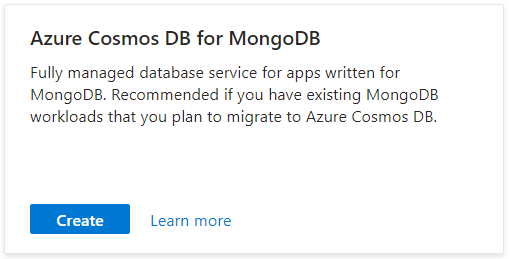
Description automatically generated

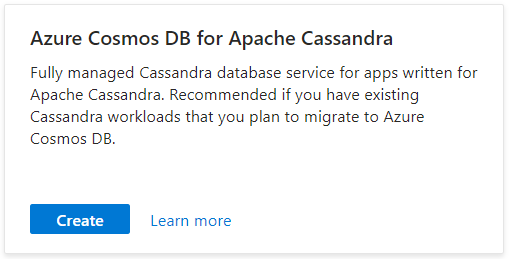
A screenshot of a computer

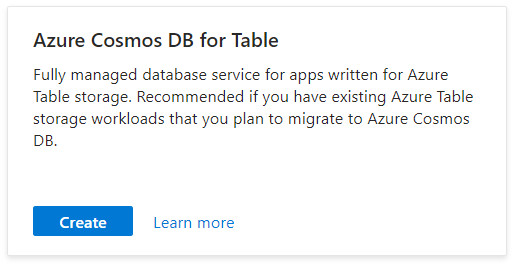
Description automatically generated

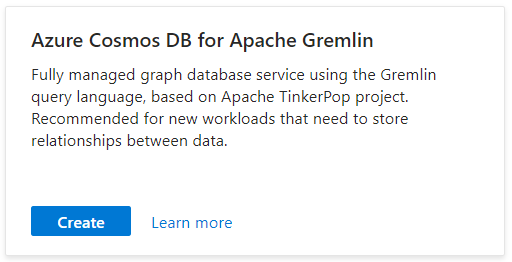












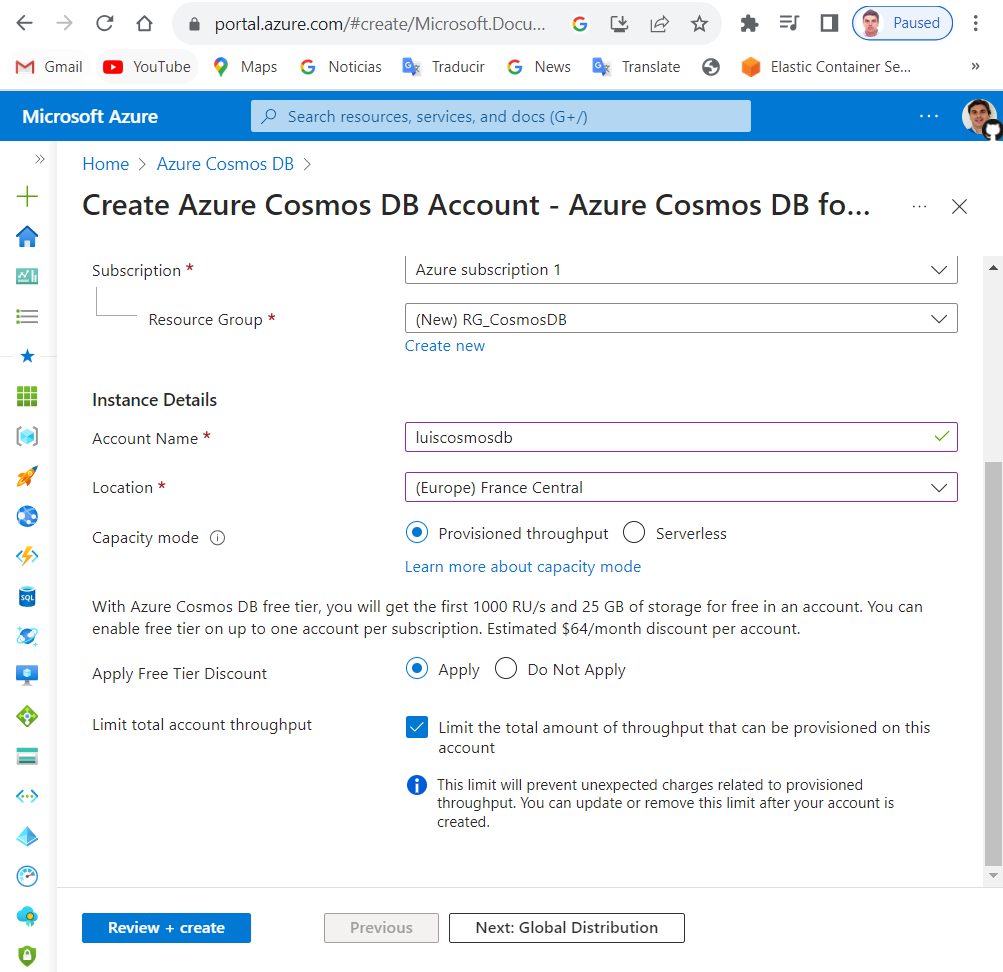


A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



A screenshot of a computer

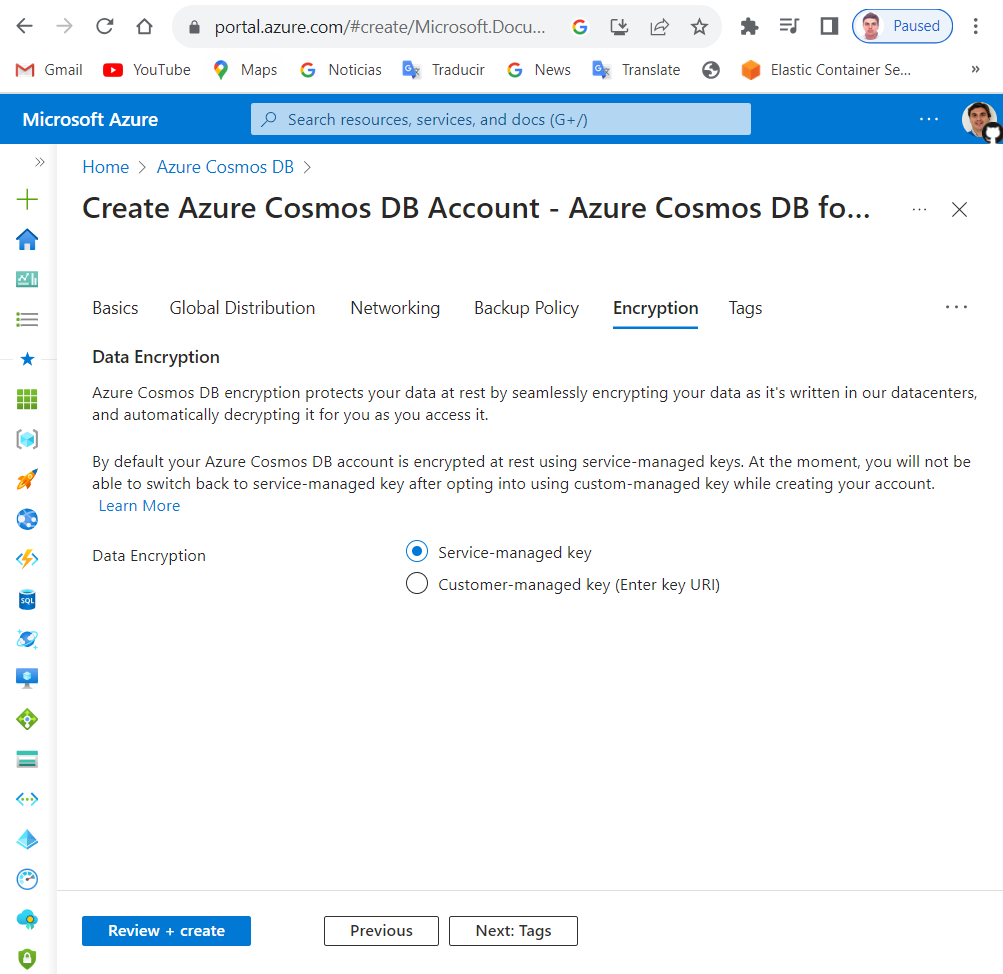
Description automatically generated

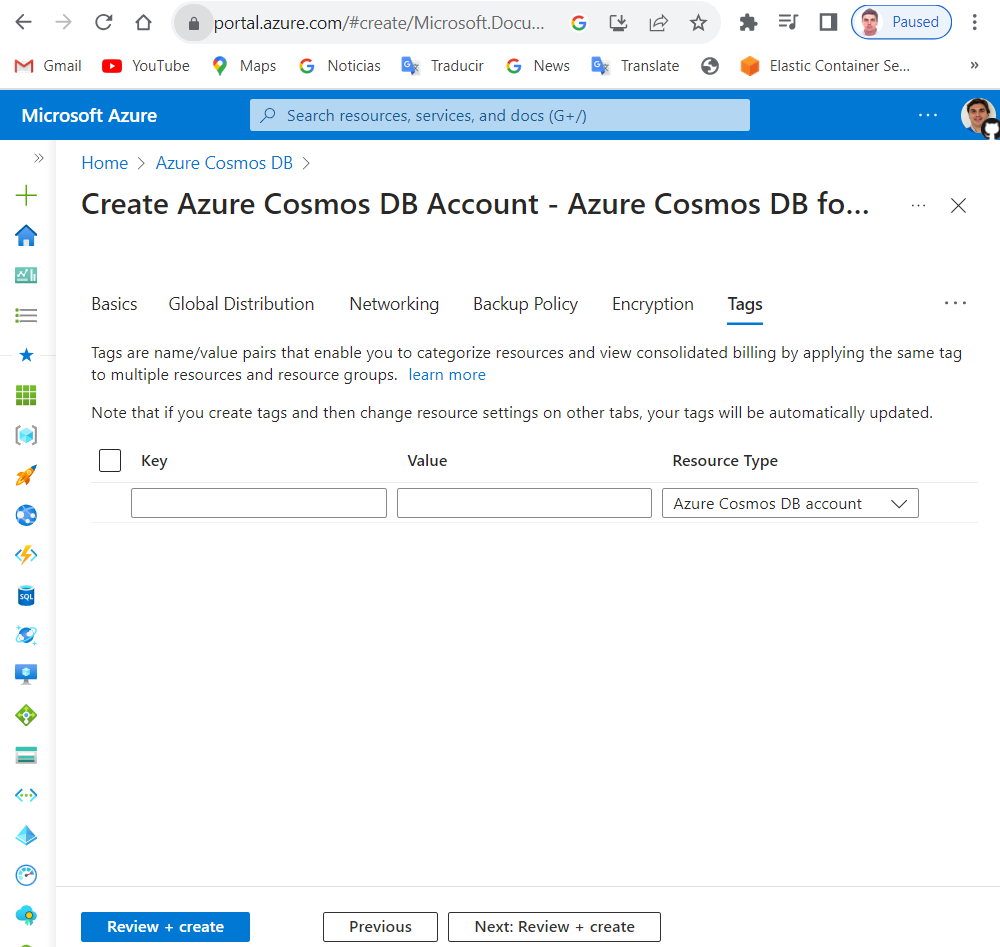
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated





A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

------------------------------------------------------------------------------------------------------------------------------------------

**.NET Sample**

------------------------------------------------------------------------------------------------------------------------------------------

A screenshot of a computer

Description automatically generated

A screenshot of a computer

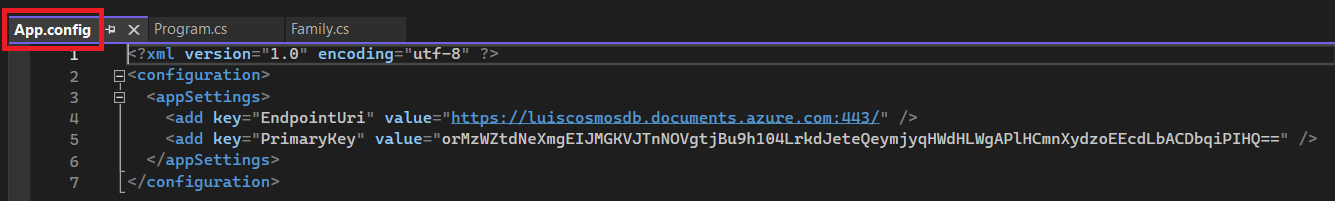
Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

Now we run the .Net sample application

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

This is the .NET sample application source code

**Program.cs**

using System;

using System.Threading.Tasks;

using System.Configuration;

using System.Collections.Generic;

using System.Net;

using Microsoft.Azure.Cosmos;

namespace CosmosGettingStartedTutorial

{

class Program

{

// The Azure Cosmos DB endpoint for running this sample.

private static readonly string EndpointUri = ConfigurationManager.AppSettings["EndPointUri"];

// The primary key for the Azure Cosmos account.

private static readonly string PrimaryKey = ConfigurationManager.AppSettings["PrimaryKey"];

// The Cosmos client instance

private CosmosClient cosmosClient;

// The database we will create

private Database database;

// The container we will create.

private Container container;

// The name of the database and container we will create

private string databaseId = "ToDoList";

private string containerId = "Items";

// <Main>

public static async Task **Main**(string[] args)

{

try

{

Console.WriteLine("Beginning operations...\n");

Program p = new Program();

await p.GetStartedDemoAsync();

}

catch (CosmosException de)

{

Exception baseException = de.GetBaseException();

Console.WriteLine("{0} error occurred: {1}", de.StatusCode, de);

}

catch (Exception e)

{

Console.WriteLine("Error: {0}", e);

}

finally

{

Console.WriteLine("End of demo, press any key to exit.");

Console.ReadKey();

}

}

// </Main>

// <GetStartedDemoAsync>

/// <summary>

/// Entry point to call methods that operate on Azure Cosmos DB resources in this sample

/// </summary>

public async Task GetStartedDemoAsync()

{

// Create a new instance of the Cosmos Client

this.cosmosClient = new CosmosClient(EndpointUri, PrimaryKey, new CosmosClientOptions() { ApplicationName = "CosmosDBDotnetQuickstart" });

await this.CreateDatabaseAsync();

await this.CreateContainerAsync();

await this.ScaleContainerAsync();

await this.AddItemsToContainerAsync();

await this.QueryItemsAsync();

await this.ReplaceFamilyItemAsync();

await this.DeleteFamilyItemAsync();

await this.DeleteDatabaseAndCleanupAsync();

}

// </GetStartedDemoAsync>

// <CreateDatabaseAsync>

/// <summary>

/// Create the database if it does not exist

/// </summary>

private async Task CreateDatabaseAsync()

{

// Create a new database

this.database = await this.cosmosClient.CreateDatabaseIfNotExistsAsync(databaseId);

Console.WriteLine("Created Database: {0}\n", this.database.Id);

}

// </CreateDatabaseAsync>

// <CreateContainerAsync>

/// <summary>

/// Create the container if it does not exist.

/// Specifiy "/partitionKey" as the partition key path since we're storing family information, to ensure good distribution of requests and storage.

/// </summary>

/// <returns></returns>

private async Task CreateContainerAsync()

{

// Create a new container

this.container = await this.database.CreateContainerIfNotExistsAsync(containerId, "/partitionKey");

Console.WriteLine("Created Container: {0}\n", this.container.Id);

}

// </CreateContainerAsync>

// <ScaleContainerAsync>

/// <summary>

/// Scale the throughput provisioned on an existing Container.

/// You can scale the throughput (RU/s) of your container up and down to meet the needs of the workload. Learn more: https://aka.ms/cosmos-request-units

/// </summary>

/// <returns></returns>

private async Task ScaleContainerAsync()

{

// Read the current throughput

try

{

int? throughput = await this.container.ReadThroughputAsync();

if (throughput.HasValue)

{

Console.WriteLine("Current provisioned throughput : {0}\n", throughput.Value);

int newThroughput = throughput.Value + 100;

// Update throughput

await this.container.ReplaceThroughputAsync(newThroughput);

Console.WriteLine("New provisioned throughput : {0}\n", newThroughput);

}

}

catch (CosmosException cosmosException) when (cosmosException.StatusCode == HttpStatusCode.BadRequest)

{

Console.WriteLine("Cannot read container throuthput.");

Console.WriteLine(cosmosException.ResponseBody);

}

}

// </ScaleContainerAsync>

// <AddItemsToContainerAsync>

/// <summary>

/// Add Family items to the container

/// </summary>

private async Task AddItemsToContainerAsync()

{

// Create a family object for the Andersen family

Family andersenFamily = new Family

{

Id = "Andersen.1",

PartitionKey = "Andersen",

LastName = "Andersen",

Parents = new Parent[]

{

new Parent { FirstName = "Thomas" },

new Parent { FirstName = "Mary Kay" }

},

Children = new Child[]

{

new Child

{

FirstName = "Henriette Thaulow",

Gender = "female",

Grade = 5,

Pets = new Pet[]

{

new Pet { GivenName = "Fluffy" }

}

}

},

Address = new Address { State = "WA", County = "King", City = "Seattle" },

IsRegistered = false

};

try

{

// Read the item to see if it exists.

ItemResponse<Family> andersenFamilyResponse = await this.container.ReadItemAsync<Family>(andersenFamily.Id, new PartitionKey(andersenFamily.PartitionKey));

Console.WriteLine("Item in database with id: {0} already exists\n", andersenFamilyResponse.Resource.Id);

}

catch(CosmosException ex) when (ex.StatusCode == HttpStatusCode.NotFound)

{

// Create an item in the container representing the Andersen family. Note we provide the value of the partition key for this item, which is "Andersen"

ItemResponse<Family> andersenFamilyResponse = await this.container.CreateItemAsync<Family>(andersenFamily, new PartitionKey(andersenFamily.PartitionKey));

// Note that after creating the item, we can access the body of the item with the Resource property off the ItemResponse. We can also access the RequestCharge property to see the amount of RUs consumed on this request.

Console.WriteLine("Created item in database with id: {0} Operation consumed {1} RUs.\n", andersenFamilyResponse.Resource.Id, andersenFamilyResponse.RequestCharge);

}

// Create a family object for the Wakefield family

Family wakefieldFamily = new Family

{

Id = "Wakefield.7",

PartitionKey = "Wakefield",

LastName = "Wakefield",

Parents = new Parent[]

{

new Parent { FamilyName = "Wakefield", FirstName = "Robin" },

new Parent { FamilyName = "Miller", FirstName = "Ben" }

},

Children = new Child[]

{

new Child

{

FamilyName = "Merriam",

FirstName = "Jesse",

Gender = "female",

Grade = 8,

Pets = new Pet[]

{

new Pet { GivenName = "Goofy" },

new Pet { GivenName = "Shadow" }

}

},

new Child

{

FamilyName = "Miller",

FirstName = "Lisa",

Gender = "female",

Grade = 1

}

},

Address = new Address { State = "NY", County = "Manhattan", City = "NY" },

IsRegistered = true

};

try

{

// Read the item to see if it exists

ItemResponse<Family> wakefieldFamilyResponse = await this.container.ReadItemAsync<Family>(wakefieldFamily.Id, new PartitionKey(wakefieldFamily.PartitionKey));

Console.WriteLine("Item in database with id: {0} already exists\n", wakefieldFamilyResponse.Resource.Id);

}

catch(CosmosException ex) when (ex.StatusCode == HttpStatusCode.NotFound)

{

// Create an item in the container representing the Wakefield family. Note we provide the value of the partition key for this item, which is "Wakefield"

ItemResponse<Family> wakefieldFamilyResponse = await this.container.CreateItemAsync<Family>(wakefieldFamily, new PartitionKey(wakefieldFamily.PartitionKey));

// Note that after creating the item, we can access the body of the item with the Resource property off the ItemResponse. We can also access the RequestCharge property to see the amount of RUs consumed on this request.

Console.WriteLine("Created item in database with id: {0} Operation consumed {1} RUs.\n", wakefieldFamilyResponse.Resource.Id, wakefieldFamilyResponse.RequestCharge);

}

}

// </AddItemsToContainerAsync>

// <QueryItemsAsync>

/// <summary>

/// Run a query (using Azure Cosmos DB SQL syntax) against the container

/// Including the partition key value of lastName in the WHERE filter results in a more efficient query

/// </summary>

private async Task QueryItemsAsync()

{

var sqlQueryText = "SELECT \* FROM c WHERE c.PartitionKey = 'Andersen'";

Console.WriteLine("Running query: {0}\n", sqlQueryText);

QueryDefinition queryDefinition = new QueryDefinition(sqlQueryText);

FeedIterator<Family> queryResultSetIterator = this.container.GetItemQueryIterator<Family>(queryDefinition);

List<Family> families = new List<Family>();

while (queryResultSetIterator.HasMoreResults)

{

FeedResponse<Family> currentResultSet = await queryResultSetIterator.ReadNextAsync();

foreach (Family family in currentResultSet)

{

families.Add(family);

Console.WriteLine("\tRead {0}\n", family);

}

}

}

// </QueryItemsAsync>

// <ReplaceFamilyItemAsync>

/// <summary>

/// Replace an item in the container

/// </summary>

private async Task ReplaceFamilyItemAsync()

{

ItemResponse<Family> wakefieldFamilyResponse = await this.container.ReadItemAsync<Family>("Wakefield.7", new PartitionKey("Wakefield"));

var itemBody = wakefieldFamilyResponse.Resource;

// update registration status from false to true

itemBody.IsRegistered = true;

// update grade of child

itemBody.Children[0].Grade = 6;

// replace the item with the updated content

wakefieldFamilyResponse = await this.container.ReplaceItemAsync<Family>(itemBody, itemBody.Id, new PartitionKey(itemBody.PartitionKey));

Console.WriteLine("Updated Family [{0},{1}].\n \tBody is now: {2}\n", itemBody.LastName, itemBody.Id, wakefieldFamilyResponse.Resource);

}

// </ReplaceFamilyItemAsync>

// <DeleteFamilyItemAsync>

/// <summary>

/// Delete an item in the container

/// </summary>

private async Task DeleteFamilyItemAsync()

{

var partitionKeyValue = "Wakefield";

var familyId = "Wakefield.7";

// Delete an item. Note we must provide the partition key value and id of the item to delete

ItemResponse<Family> wakefieldFamilyResponse = await this.container.DeleteItemAsync<Family>(familyId,new PartitionKey(partitionKeyValue));

Console.WriteLine("Deleted Family [{0},{1}]\n", partitionKeyValue, familyId);

}

// </DeleteFamilyItemAsync>

// <DeleteDatabaseAndCleanupAsync>

/// <summary>

/// Delete the database and dispose of the Cosmos Client instance

/// </summary>

private async Task DeleteDatabaseAndCleanupAsync()

{

DatabaseResponse databaseResourceResponse = await this.database.DeleteAsync();

// Also valid: await this.cosmosClient.Databases["FamilyDatabase"].DeleteAsync();

Console.WriteLine("Deleted Database: {0}\n", this.databaseId);

//Dispose of CosmosClient

this.cosmosClient.Dispose();

}

// </DeleteDatabaseAndCleanupAsync>

}

}

**Family.cs**

using Newtonsoft.Json;

namespace CosmosGettingStartedTutorial

{

public class Family

{

[JsonProperty(PropertyName = "id")]

public string Id { get; set; }

[JsonProperty(PropertyName = "partitionKey")]

public string PartitionKey { get; set; }

public string LastName { get; set; }

public Parent[] Parents { get; set; }

public Child[] Children { get; set; }

public Address Address { get; set; }

public bool IsRegistered { get; set; }

public override string ToString()

{

return JsonConvert.SerializeObject(this);

}

}

public class Parent

{

public string FamilyName { get; set; }

public string FirstName { get; set; }

}

public class Child

{

public string FamilyName { get; set; }

public string FirstName { get; set; }

public string Gender { get; set; }

public int Grade { get; set; }

public Pet[] Pets { get; set; }

}

public class Pet

{

public string GivenName { get; set; }

}

public class Address

{

public string State { get; set; }

public string County { get; set; }

public string City { get; set; }

}

}

------------------------------------------------------------------------------------------------------------------------------------------

**JAVA Sample**

------------------------------------------------------------------------------------------------------------------------------------------