Microsoft

Connect to Azure Cosmos DB for NoSQL with the SDK

Agenda

- Import and use the Azure Cosmos DB for NoSQL SDK
- Configure the Azure Cosmos DB for NoSQL SDK

Import and use the Azure Cosmos DB for NoSQL SDK

Understand the SDK

Class	Description
Microsoft.Azure.Cosmos.**CosmosClient**	Client-side logical representation of an Azure Cosmos DB account and the primary class used for the SDK
Microsoft.Azure.Cosmos.**Database**	Logically represents a database client-side and includes common operations for database management
Microsoft.Azure.Cosmos.**Container**	Logically represents a container client-side and includes common operations for container management

The *Microsoft.Azure.Cosmos* library is the latest version of the .NET SDK for Azure Cosmos DB for NoSQL.

Import from a package manager

The *Microsoft.Azure.Cosmos* library, including all of its previous versions, are hosted on *nuget* to make it easier to import the library into a .NET application.

Importing a NuGet package

```
dotnet add package
Microsoft.Azure.Cosmos
```

```
dotnet add package
Microsoft.Azure.Cosmos \
    --version 3.22.1
```

.NET project file

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net6.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="Microsoft.Azure.Cosmos"
Version="3.22.1" />
    </ItemGroup>
</Project>
```

Connect to an online account

Once the *Microsoft.Azure.Cosmos* library is imported, you can begin using the namespaces and classes within your .NET project.

Import the namespace

```
using Microsoft.Azure.Cosmos;
```

Use the CosmosClient class

```
// Use with a connection string
string connectionString = "AccountEndpoint=https-
://dp420.documents.azure.com:443/;AccountKey=fDR2ci9QgkdkvERTQ==";

CosmosClient client = new (connectionString);
```

```
// Use with an endpoint and key
string endpoint = "https-
://dp420.documents.azure.com:443/";
string key = "fDR2ci9QgkdkvERTQ==";

CosmosClient client = new (endpoint, key);
```

Read properties of the account

```
AccountProperties account = await
client.ReadAccountAsync();
```

Property	Description
Id	Gets the unique name of the account
ReadableRegions	Gets a list of readable locations for the account
WritableRegions	Gets a list of writable locations for the account
Consistency	Gets the default consistency level for the account

Interact with a database and a container

Once you have a client instance, you can retrieve or create a database or container.

Interact with a database

```
// Retrieve an existing database
Database database =
client.GetDatabase("cosmicworks");
```

```
// Create a new database
Database database = await
client.CreateDatabaseAsync("cosmicworks");
```

```
// Create database if it doesn't already exist
Database database = await
client.CreateDatabaseIfNotExistsAsync("cosmicwork
s");
```

Interact with a container

```
// Retrieve an existing container
Container container = database.GetContainer("products");
```

```
// Create a new container
Container container = await
database.CreateContainerAsync(
    "cosmicworks",
    "/categoryId",
    400
);
```

```
// Create container if it doesn't already exist
Container container = await
database.CreateContainerIfNotExistsAsync(
    "cosmicworks",
    "/categoryId",
    400
);
```

Implement client singleton

CosmosClient class features implemented on your behalf:

- Instances are already thread-safe
- Instances efficiently manage connections
- Instances cache addresses when operating in direct mode

** The Azure Cosmos DB for NoSQL SDK team recommends that you use a single instance per AppDomain for the lifetime of the application.*

Configure connectivity mode

The *CosmosClientOptions* class provides a range of options that you can configure for the client when it connects to an account.

Overriding default client options

```
// Constructor that takes in an endpoint and key
CosmosClientOptions options = new ();
CosmosClient client = new (endpoint, key, options);
```

```
//  Constructor that takes the connection string
CosmosClientOptions options = new ();
CosmosClient client = new (connectionString, options);
```

Changing the connection mode

```
// Configures the client to use Direct connection mode.
CosmosClientOptions options = new ()
{ ConnectionMode = ConnectionMode.Direct };
```

```
// Configures the client to use Gateway connection mode.
CosmosClientOptions options = new ()
{ ConnectionMode = ConnectionMode.Gateway };
```

Setting the preferred application region[s]

```
// Configs single preferred region for client to connect to.
CosmosClientOptions options = new ()
{ApplicationRegion = "westus" };
```

```
// Configs the client to use custom failover/priority list.
CosmosClientOptions options = new ()
{ApplicationPreferredRegions = new List<string> { "westus", "eastus"
} };
```

Changing the current consistency level

```
// Configures the client to use eventual consistency.
CosmosClientOptions options = new ()
{ConsistencyLevel = ConsistencyLevel.Eventual };
```

Consistency Levels

- Bounded Staleness
- ConsistentPrefix
- Eventual
- Session
- Strong

Lab – Connect to Azure Cosmos DB for NoSQL with the SDK

- Prepare your development environment
- Create an Azure Cosmos DB for NoSQL account
- View the Microsoft.Azure.Cosmos library on NuGet
- Import the Microsoft.Azure.Cosmos library into a .NET project
- Use the Microsoft.Azure.Cosmos library
- Test the script

Configure the Azure Cosmos DB for NoSQL SDK

Enable offline development

Azure Cosmos DB emulator

- The emulator is available to run in Windows, Linux, or as a Docker container image.
- To pull the Azure Cosmos DB emulator Docker Image run the CLI command:

```
docker pull mcr.microsoft.com/cosmosdb/linux/azure-cosmos-emulator
```

Configuring the SDK to connect to the emulator

```
// The emulator's endpoint is https://localhost:<port>/ using SSL with the default port set to 8081.
string endpoint = "https://localhost:8081/";

// The emulator's key is a static well-known authentication key. The default value for this key is:
string key = "C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XIw/Jw==";

// Once those variables are set, create the CosmosClient like you would for a cloud-based account.
CosmosClient client = new (endpoint, key);
```

Handle connection errors

Since requests could fail for various reasons, you should have error handling logic.

- The Azure Cosmos DB for NoSQL SDK for .NET has built-in logic to handle common transient failures for read and query requests.
- The SDK does NOT automatically retry write requests. Your application code should implement retry logic for failed writes.

Common HTTP status codes where retrying your request makes sense.

- 429: Too many requests
- 449: Concurrency error
- 500: Unexpected service error
- 503: Service unavailable

Common HTTP status codes that also might need error handling.

- 400: Bad request
- 401: Not authorized
- 403: Forbidden
- 404: Not found

Implement threading and parallelism

The SDK implements thread-safe types and some degrees of parallelism.

Use async/await in .NET

```
// Use Task-based features to asynchronously invoke SDK client
methods.
Database database = await
client.CreateDatabaseIfNotExistsAsync("cosmicworks");
```

```
//  Avoid blocking the async exec using Task.Wait or
Task.Result like:
Database database =
client.CreateDatabaseIfNotExistsAsync("cosmicworks").Result;
```

Configure max concurrency, parallelism, and buffered item count

```
// The query returns 500 items per page.
QueryRequestOptions options = new () { MaxItemCount = 500 };
```

```
// The query runs 5 concurrent operations on the client-side.
QueryRequestOptions options = new () { MaxConcurrency = 5 };
```

```
// The Query buffers 5000 items at the client-side.
QueryRequestOptions options = new () { MaxBufferedItemCount =
5000 };
```

Use built-in iterators instead of LINQ methods

```
// Use SDK included methods such as ToFeedIterator<T> that
// asynchronously retrieves the results and don't block other
calls.
container.GetItemLinqQueryable<T>()
    .Where(i => i.categoryId == 2)
    .ToFeedIterator<T>();
```

```
// Avoid LINQ methods such as ToList that block other calls.
container.GetItemLinqQueryable<T>()
    .Where(i => i.categoryId == 2)
    .ToList<T>();
```

Avoid client-side resource-related timeouts

Issues at the client machines that cause request timeouts

- High CPU Utilization
- High Port Utilization

Configure logging

The SDK includes a client builder class that simplifies the process of injecting custom handlers into the HTTP requests and responses.

Client builder

```
// To use the builder, add the using directive Microsoft.Azure.Cosmos.Fluent.
using Microsoft.Azure.Cosmos.Fluent;

// Create an instance with either the connection string or endpoint/key.
CosmosClientBuilder builder = new (connectionString);
CosmosClientBuilder builder = new (endpoint, key);

// Add the fluent methods and then build the CosmosClient instance.
CosmosClient client = builder.Build();
```

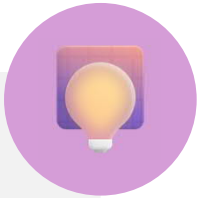
Creating a custom RequestHandler implementation

```
public class LogHandler : RequestHandler
{
    public override async Task<ResponseMessage> SendAsync(RequestMessage request, CancellationToken cancellationToken)
    {
        Console.WriteLine($"[{request.Method.Method}]\t{request.RequestUri}");
        ResponseMessage response = await base.SendAsync(request, cancellationToken);
        Console.WriteLine($"[{Convert.ToInt32(response.StatusCode)}]\t{response.StatusCode}");
        return response;
    }
}
```

Lab – Configure the Azure Cosmos DB for NoSQL SDK for offline development

- Prepare your development environment
- Start the Azure Cosmos DB Emulator
- Connect to the emulator from the SDK
- View the changes in the emulator
- Create and view a new container
- Stop the Azure Cosmos DB Emulator

Review



1. Which class should you use to interact with containers in the Azure Cosmos DB SDK for .NET?

   - ❑ DocumentCollection.
   - ❑ CosmosContainer.
   - ☑ Container.

2. What is the name of the Azure Cosmos DB SDK for .NET on NuGet?

   - ☑ Microsoft.Azure.Cosmos.
   - ❑ Microsoft.Azure.Documents.
   - ❑ Microsoft.Azure.DocumentDB.

3. Which class should you inherit from to create a class that intercepts SDK-side HTTP requests and inject extra logic?

   - ☑ RequestHandler
   - ❑ HttpRequest
   - ❑ RequestMessage