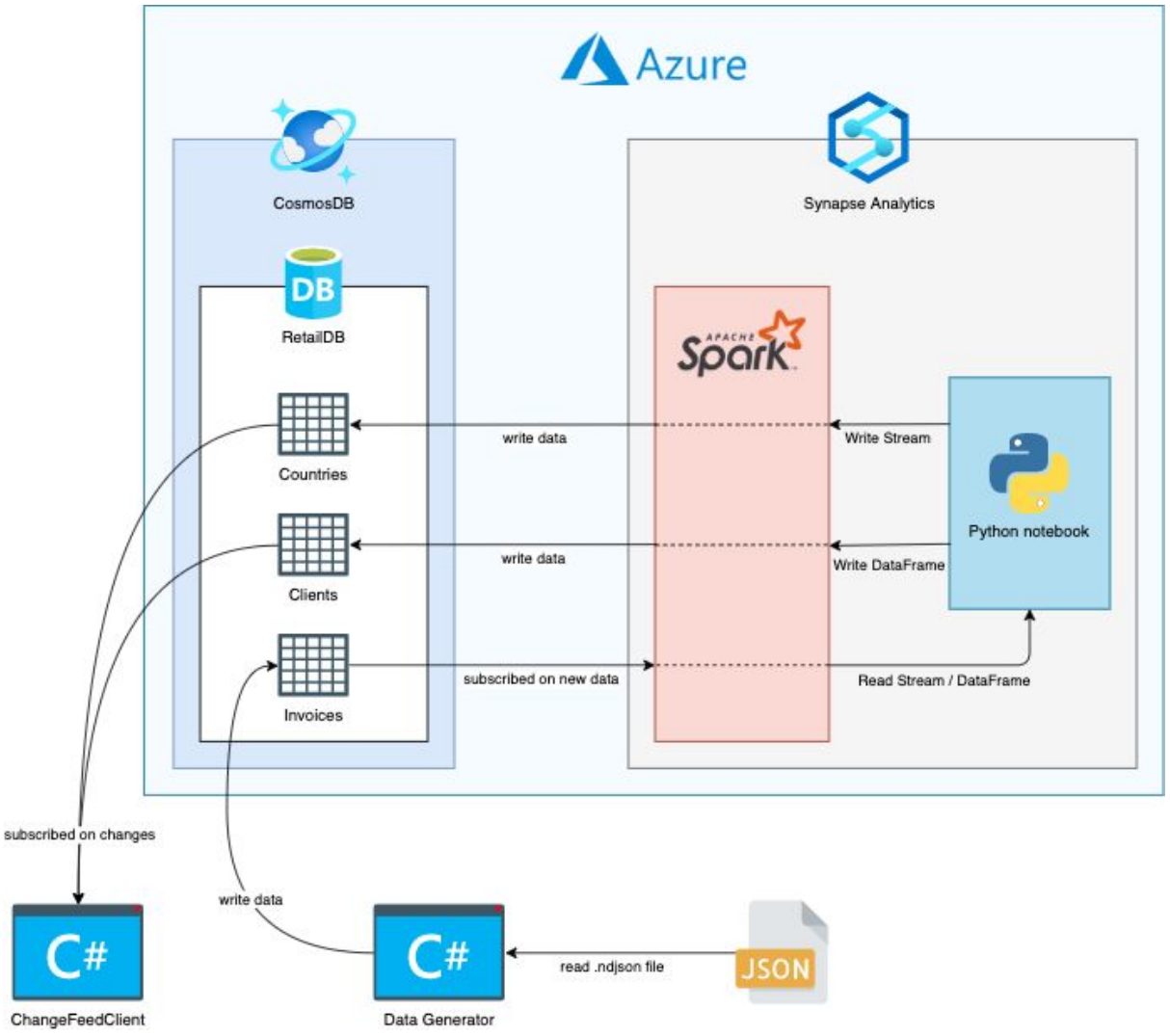
Azure Cosmos DB Real-time Data Analysis Tutorial

Kasper Kądzielawa, Szymon Wieczorek

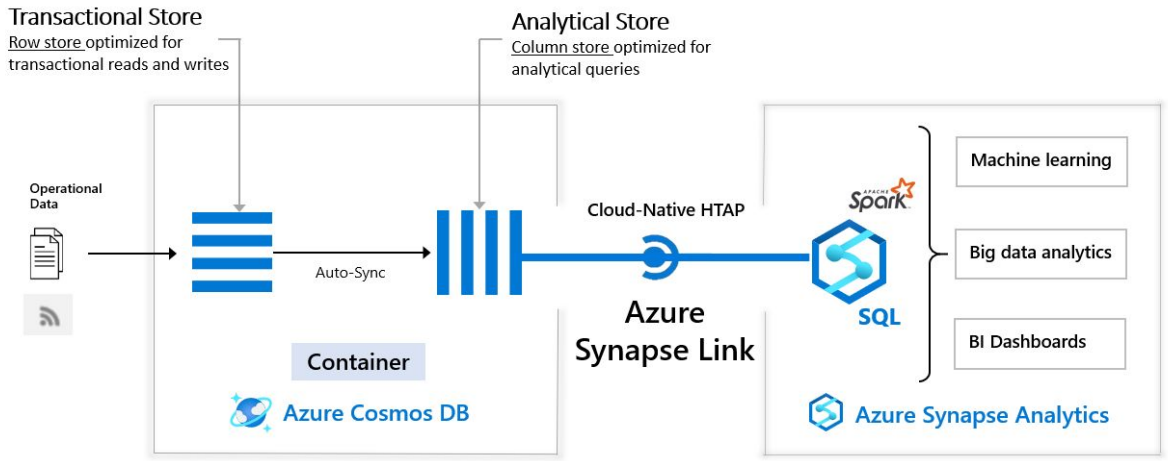
Introduction

Our dataset is a set of invoice document records. Invoices have their own **id**, **date**, a list of **items**, as well as **client** and **country** identification.

We will perform batch and real-time aggregation of data ingested into CosmosDB. Using Azure Synapse Analytics and Spark to aggregate dataframes and change feed streams, then inserting (or streaming) the data back to CosmosDB.



Analytical Store and integration with Azure Synapse Analytics



Useful documentation

- CosmosDB Documentation
- Change Feed
- Analytical Store
- Change Feed design patterns
- Change Feed processor
- Azure Synapse Link for CosmosDB
- Synapse Link integration use cases
- Use case description that inspired this tutorial

Prerequisites

- student account / free trial account on Microsoft Azure
    - CosmosDB is free within some limits that we will not exceed
    - Azure Synapse Link doesn't have any free tier + it is expensive (it can cost approx 50 PLN for 4h operation of the smallest cluster consisting on the cheapest nodes)
- .NET Core 5.0 SDK to build and run auxiliary programs

Environment preparation

In the repository, we have two command line utilities that will help us with our task. Both of them connect to CosmosDB and create appropriate artifacts (database & containers).

Clone the repository & build the solution

1. Ensure you have .NET 5.0 SDK installed
2. Clone git repository: https://github.com/kaskadz/azure-cosmos-db-spark-tutorial.git
3. Go to the repo's root directory and run **dotnet build** to build the solution.

Data Generator

DataGenerator is a utility that takes the data form the ndjson file and inserts the records into the invoices container.

```
1 usage  Kasper
private static async Task Execute(ICosmosService cosmosService, DataSupplier dataSupplier, int skip, int take)
{
    var retailDb:Database = await cosmosService.CreateDatabaseAsync("RetailDb");
    var transactionsContainer = await cosmosService.CreateContainerAsync(retailDb, containerId: "invoices",
        TransactionElementEntry.PartitionKeyPath); // Task<Container>

    await dataSupplier.IngestData(transactionsContainer, skip, take);
}
```

```
1 usage  Kasper
public async Task IngestData(Container container, int skip, int take)
{
    IAsyncEnumerable<TransactionElementEntry> transactionsAsyncEnumerable = _ingestDataReader // IngestDataReader
        .GetTransactionsAsyncEnumerable()
        .Skip(skip)
        .Take(take);

    Console.OutputEncoding = System.Text.Encoding.UTF8;
    using var progress = new ProgressBar(
        maxTicks: take,
        message: $"Documents progress",
        options: ProgressBarOptions);

    var stopwatch = Stopwatch.StartNew();
    await foreach (var transactionElementEntry in transactionsAsyncEnumerable)
    {
        await _cosmosService.AddItemToContainerAsync(container, transactionElementEntry);
        progress.Tick();
        progress.EstimatedDuration = CalculateEstimatedTimespan(take, stopwatch, progress);
    }
    stopwatch.Stop();
}
```

Change Feed Client

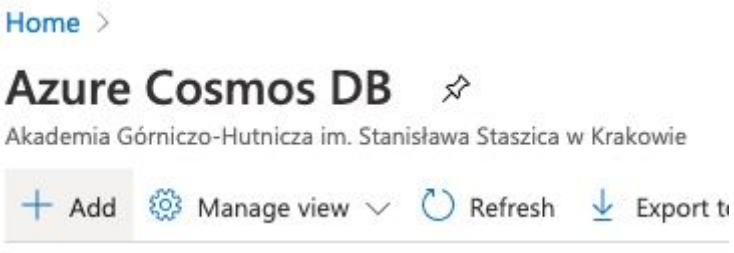
ChangeFeedClient is a utility that creates a **ChangeFeedProcessor** and subscribes to the change feed of the **countries** or **clients** container. You can quit this program by hitting the **q** button.

```
2 usages  Kasper *
public static async Task<IChangeFeedWriter> InitializeAsync(ConnectionOptions connectionOptions,
    string databaseName,
    string containerName,
    string partitionKeyPath,
    Container.ChangesHandler<T> changesHandler)
{
    var cosmosClient = new CosmosClient(connectionOptions.EndpointUri, connectionOptions.PrimaryKey);

    Database database = await cosmosClient.CreateDatabaseIfNotExistsAsync(databaseName);

    Container container =
        await database.CreateContainerIfNotExistsAsync(id:containerName, partitionKeyPath);

    Container leaseContainer =
        await database.CreateContainerIfNotExistsAsync(id:$"{containerName}Lease", partitionKeyPath);

    ChangeFeedProcessor changeFeedProcessor = container // Container
        .GetChangeFeedProcessorBuilder($"{containerName}ChangeFeed", changesHandler)
        .WithInstanceName("consoleProcessor")
        .WithLeaseContainer(leaseContainer) // ChangeFeedProcessorBuilder
        .Build();

    return new ChangeFeedConsoleWriter<T>(cosmosClient, changeFeedProcessor);
}
```

Infrastructure set up

Creating CosmosDB Account

1. Log in to your Azure Portal (https://portal.azure.com/)
2. Choose Azure CosmosDB
3. Add new account



4. Create new account as follows:

    Subscription: Free Trial

    Resource Group: Create new e.g. : adzd-cosmos

    Account name: e.g. adzd-account

    API: Core (SQL)

Notebooks: On

Location: East USs

Capacity mode: **Provisioned throughput**

Account Type: Non-produciton

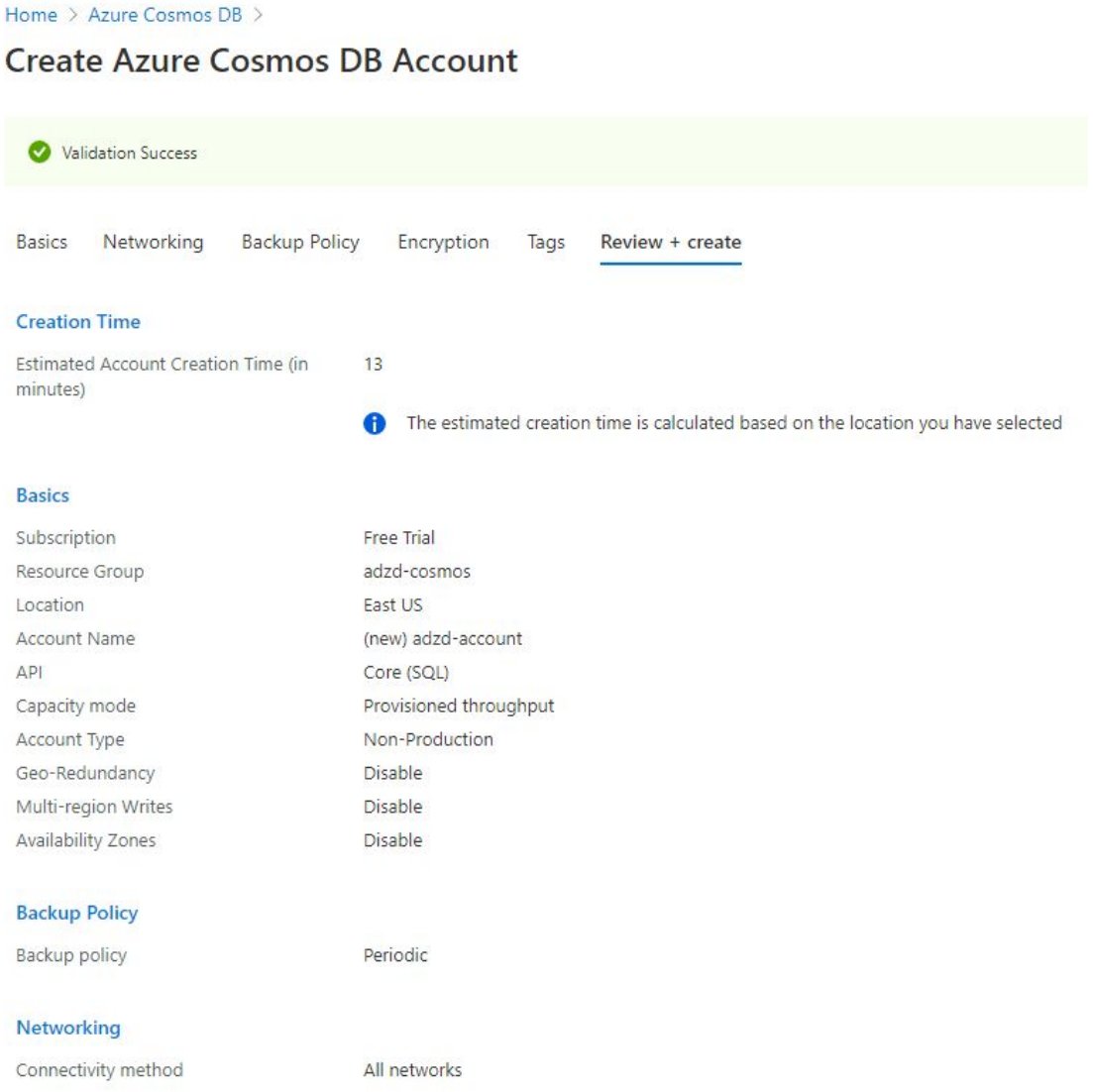
Multi-region Writes: Disable

Availability Zones: Disable

Networking:

Connectivity method: All Networks

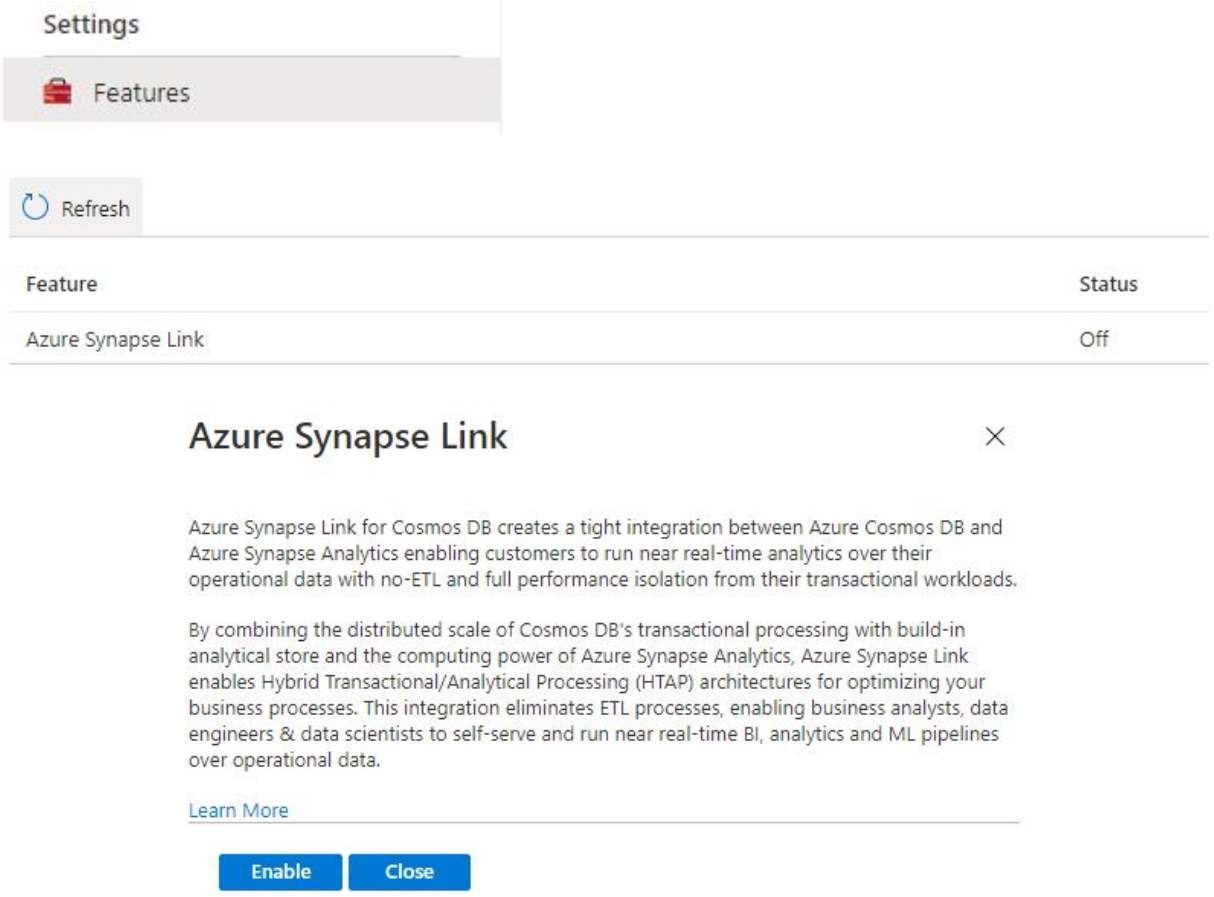
5. Click ***Review + Create***, you should see sth like below:



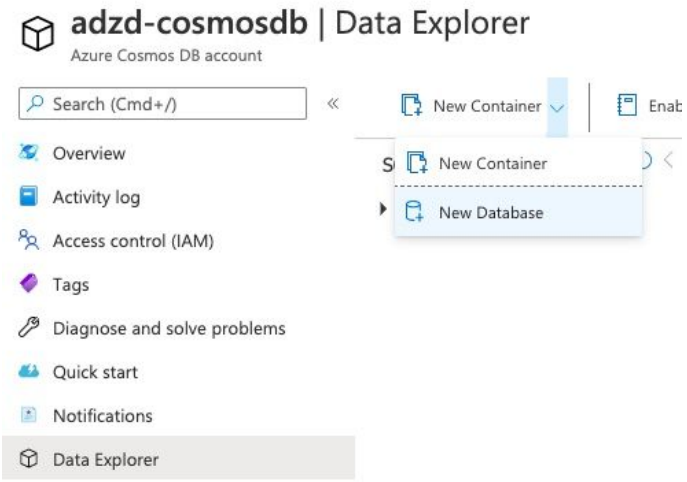
6. Click ***Create***
7. Wait for the deployment to finish

Using Analytical Storage

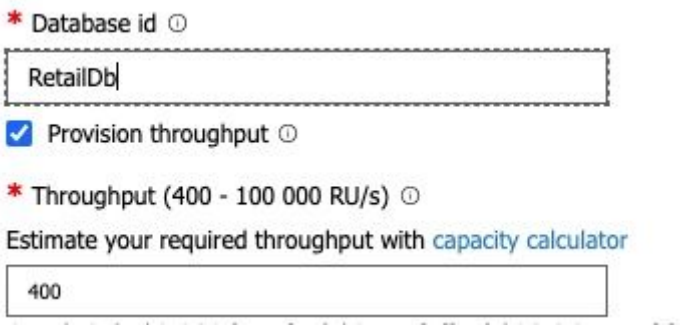
Click Settings > Features > Azure Synapse Link > Enable



8. Create ***RetailDB*** Database in CosmosDB:
    a. Go to Data Explorer
    b. Click *New Database*

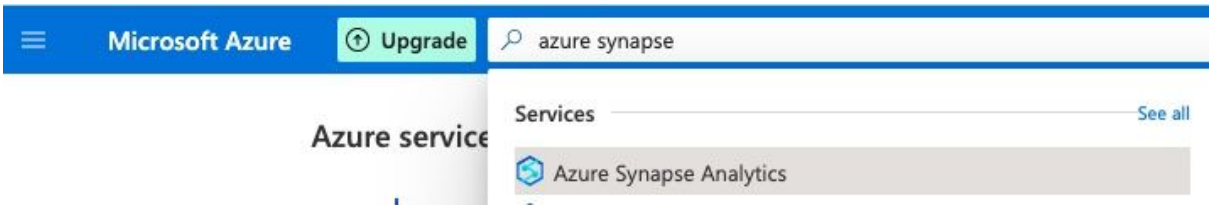


    c. Fill form as below and accept with OK button
    Database id: **RetailDb**

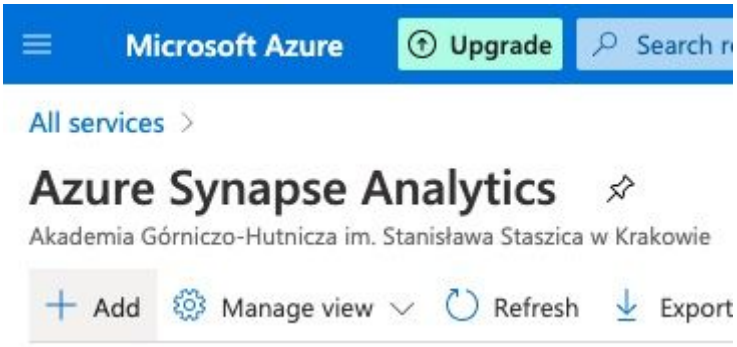

* Database id
RetailDb
Provision throughput
* Throughput (400 - 100 000 RU/s)
Estimate your required throughput with capacity calculator
400

Azure Synapse Analytics

1. Search for Azure Synapse Analytics and open it:

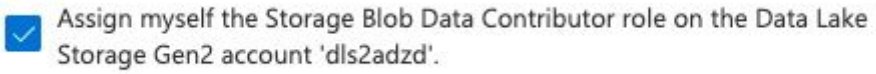


2. Click ***Add*** button to create a new workspace:

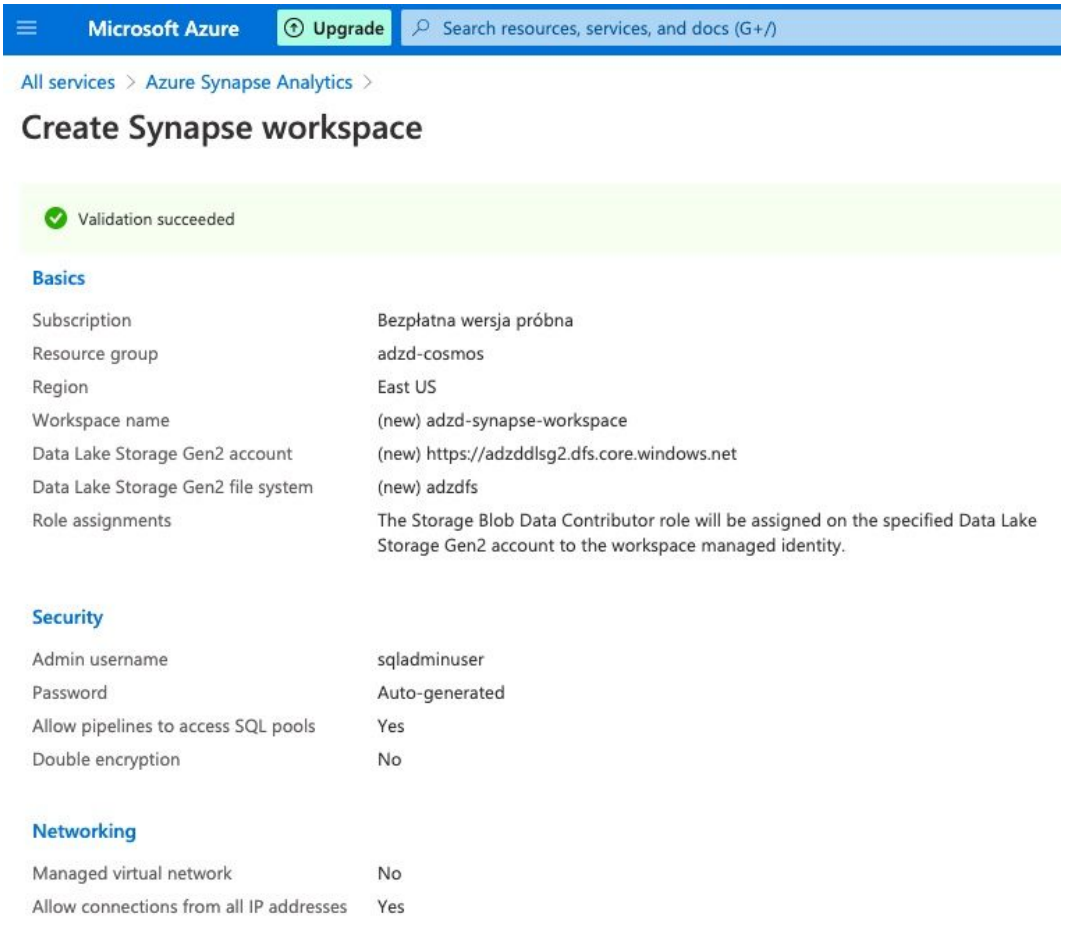


3. Create Synapse Workspace as follows:

    Subscription: Free Trial
    Resource group: adzd-cosmos (previously created when configurating cosmosdb account)
    Region: East US
    Workspace name: e.g. adzd-synapse-workspace
    Data Lake Storeage Gen2
        account: create new e.g. adzddlsg2
        file system: create new e.g. adzdfs



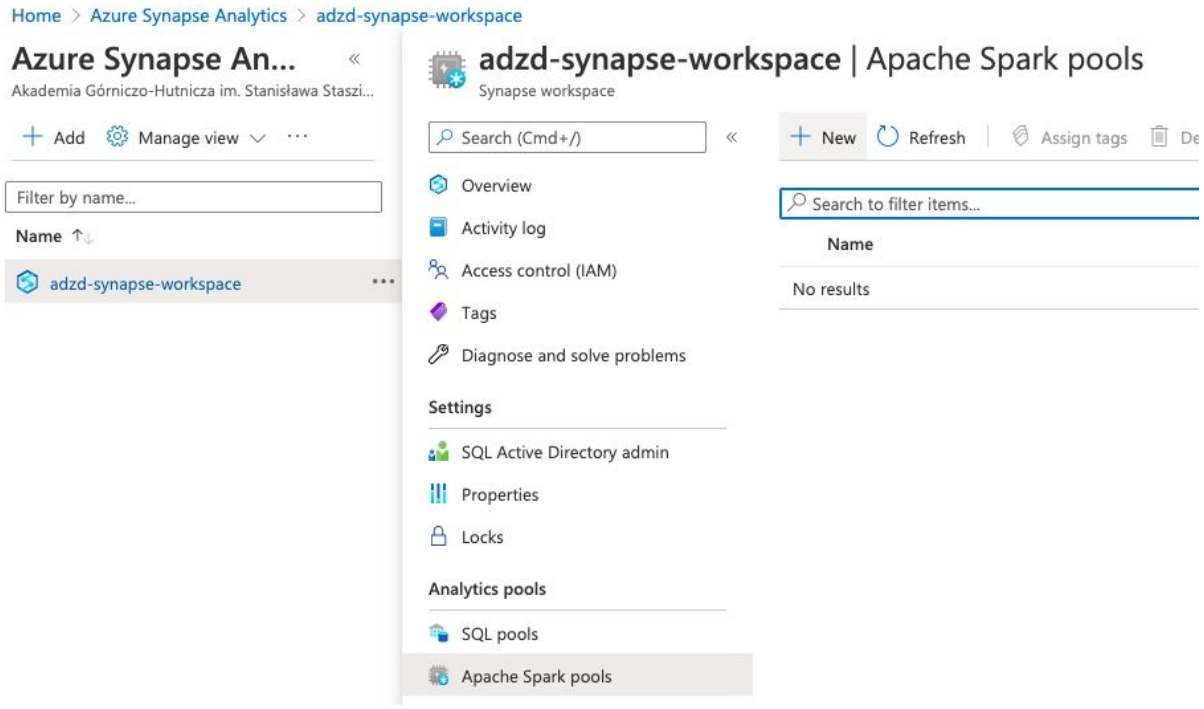
4. Click ***Review + Create***, you should see screen as below:



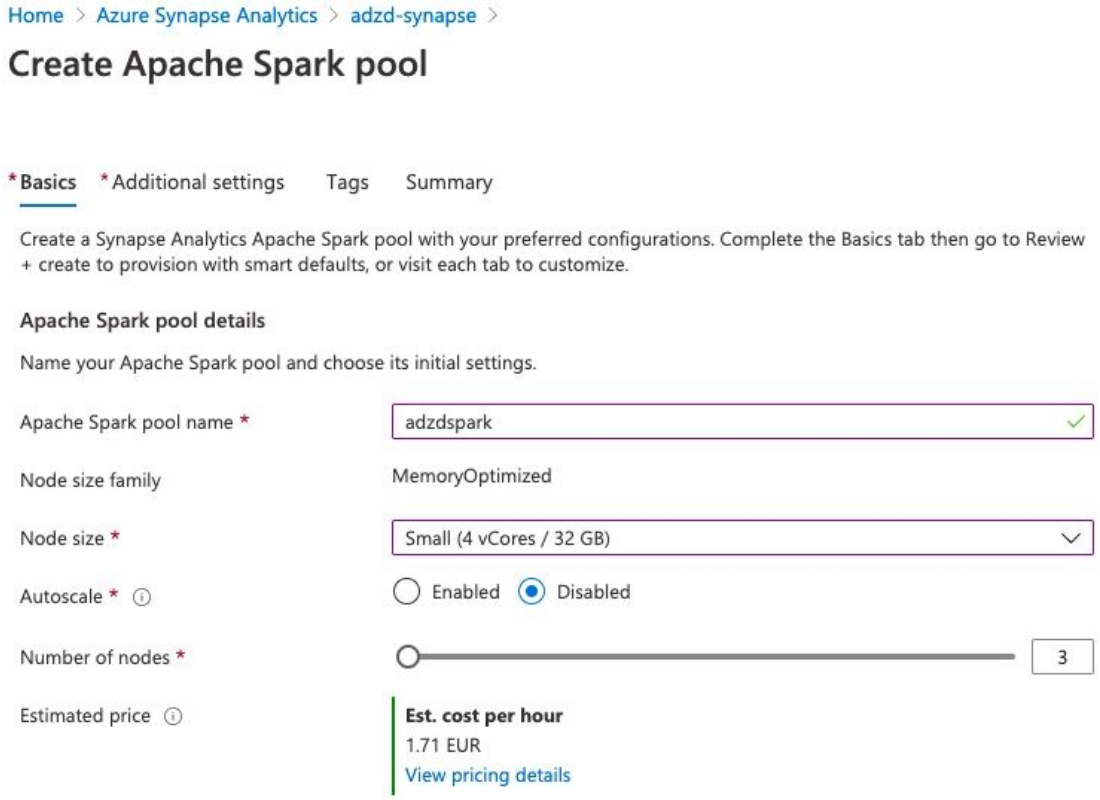
5. Select ***Create***. Your workspace is ready in a few minutes.

Creating Apache Spark Pool in Azure Synapse Analytics

1. Go to created Synapse Workspace and select **Apache Spark pools** from *Analytics pools menu*
2. Add new Spark pool by clicking **+ New**

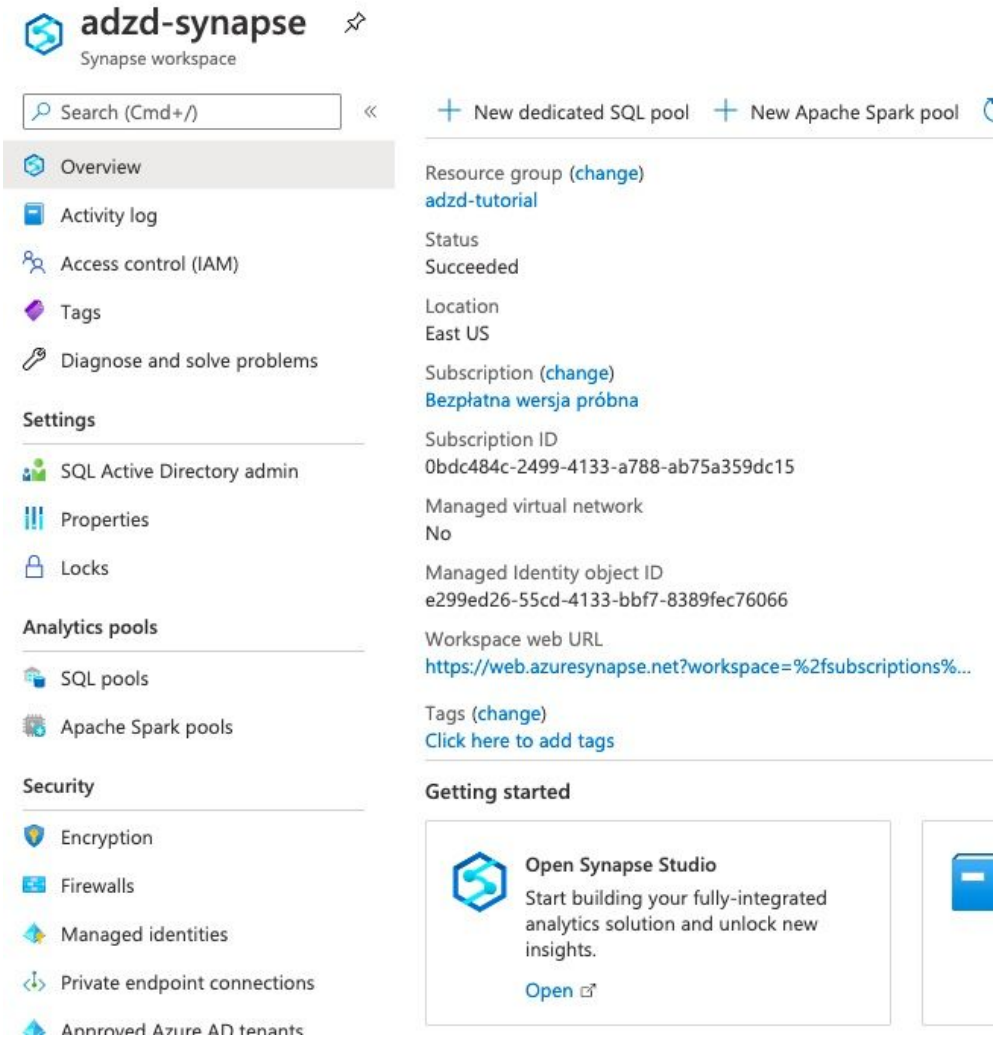


3. Create Spark pool as follows, and accept by clicking *Create*:

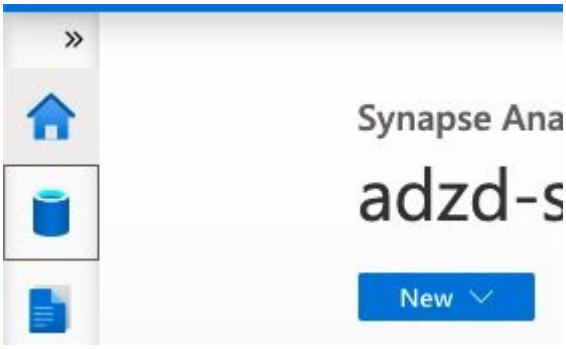


Linking CosmosDB with Synapse Analytics via Synapse Link

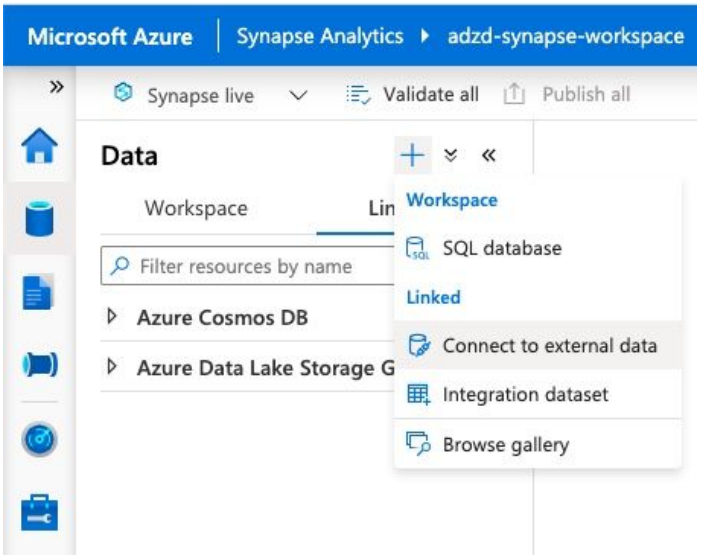
1. Go to Azure Synapse Analytics
2. Select created before Synapse Workspace
3. Click on **Workspace web URL** or **Open Synapse Studio**



4. In Synapse Studio click **Data** on right-side menu

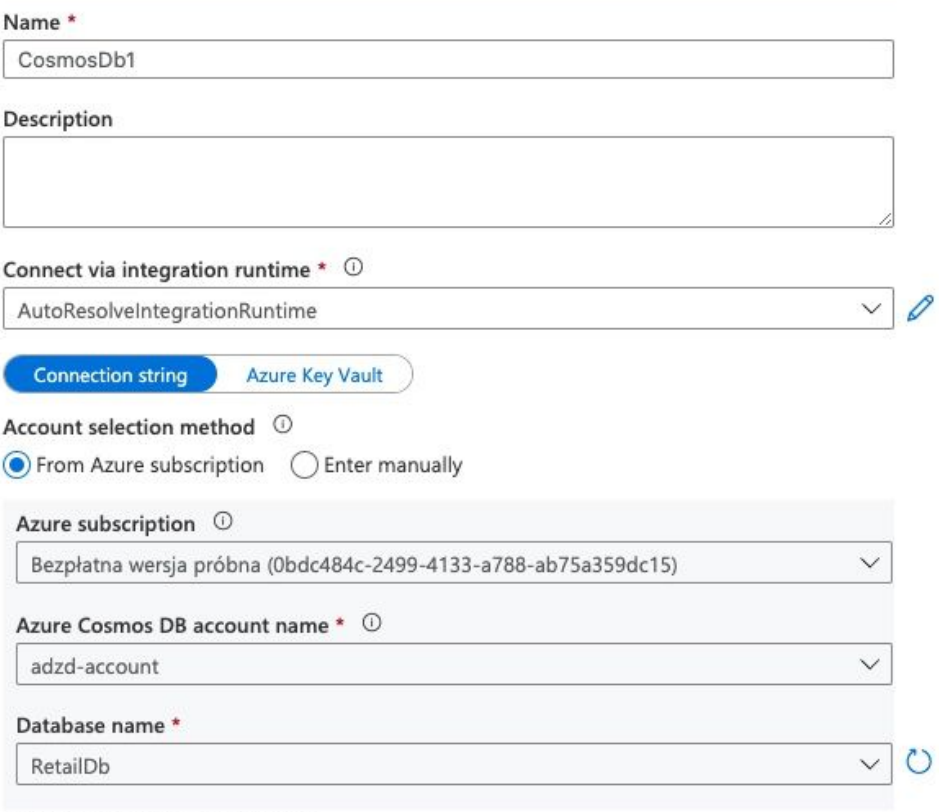


5. Click add button and ***Connect to external data***

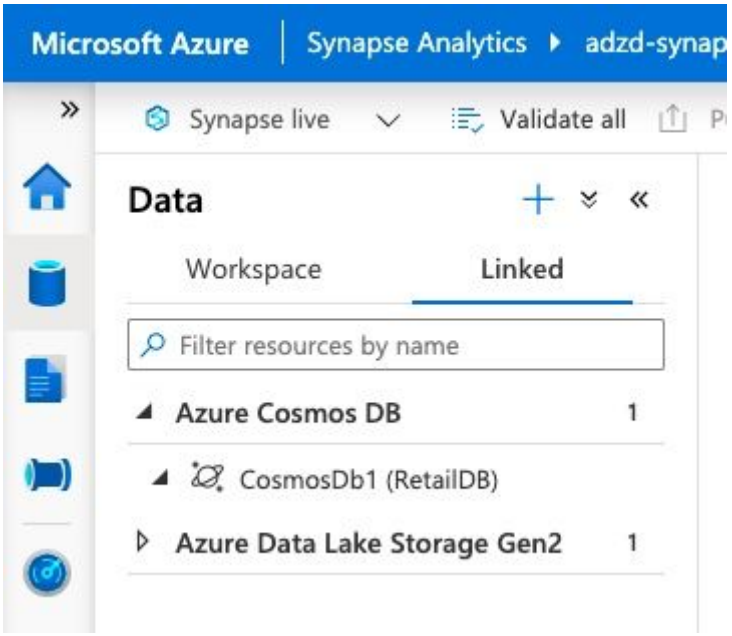


6. Choose Azure Cosmos DB (SQL API)

7. Fill form as follows and accept by clicking *create*



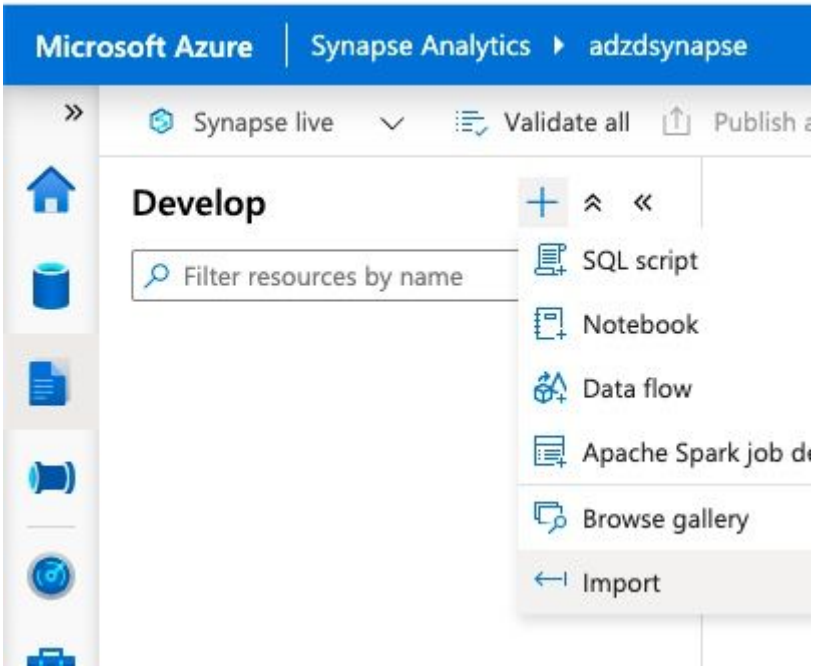
8. Refresh (F5) *Data* page, click **Linked** and you should see one connection to Azure CosmosDB - CosmosDb1 (RetailDb)



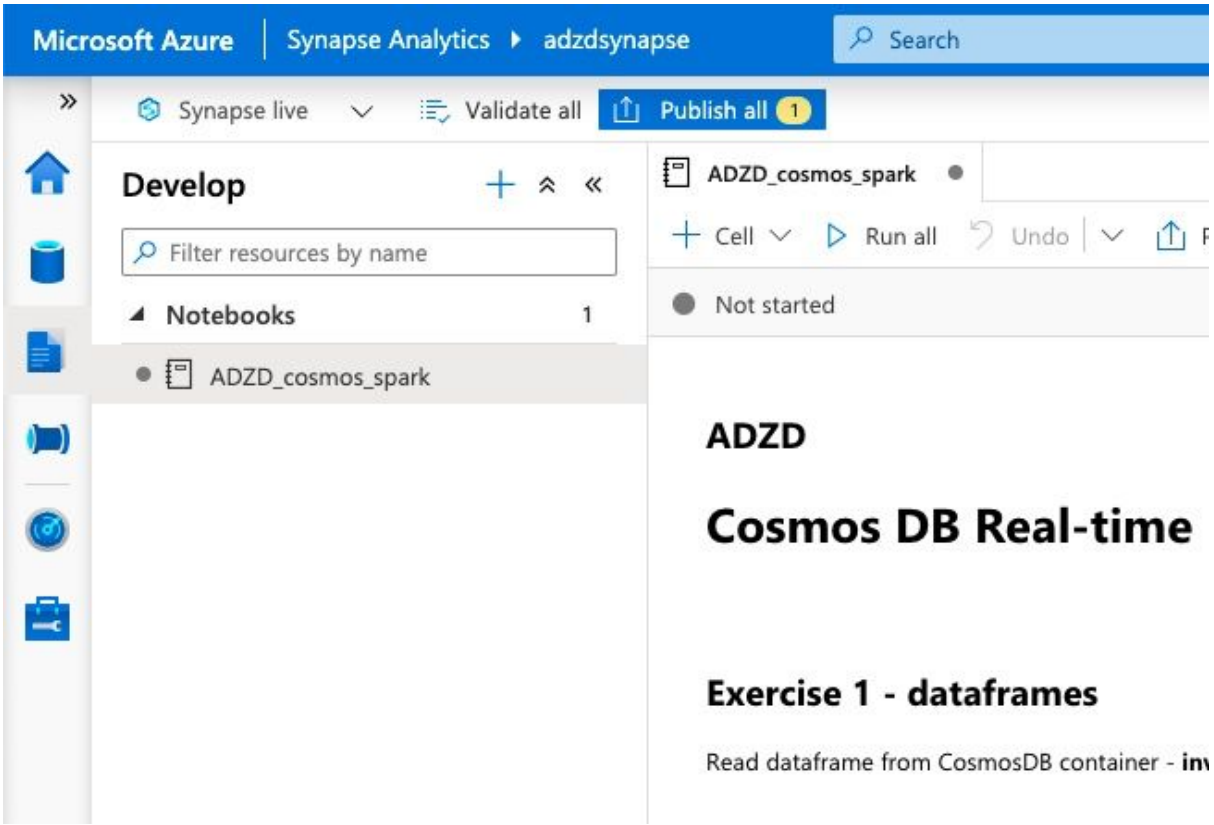
Workspace preparation

Prepare notebook for running exercises

1. In synapse studio open **Develop** tab, click **+**, and select import

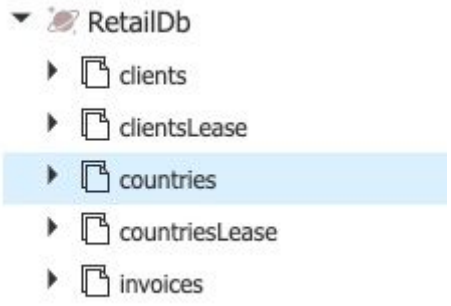


2. Import jupyter notebook ***ADZD_cosmos_spark.ipynb***



Inserting initial data

1. In the **appsettings.json** enter the endpoint and the primary key for the account you have created in the previous section. (Azure Cosmos DB > adzd-account > Keys)
2. Use dotnet run to create the container and ingest some initial data into it. Type **dotnet run -p AzureCosmosSparkTutorial.DataGenerator --File="<path-to-ndjson>" --Skip 0 --Take 500** in order to insert first 500 records to the container (replacing <path-to-ndjson> with the path of the data file from the **/data** folder in the repository)
3. Ensure that the invoices container was created. In Azure go to CosmosDB, select your account, and click DataExplorer. You should see the **invoices** container in the container list:



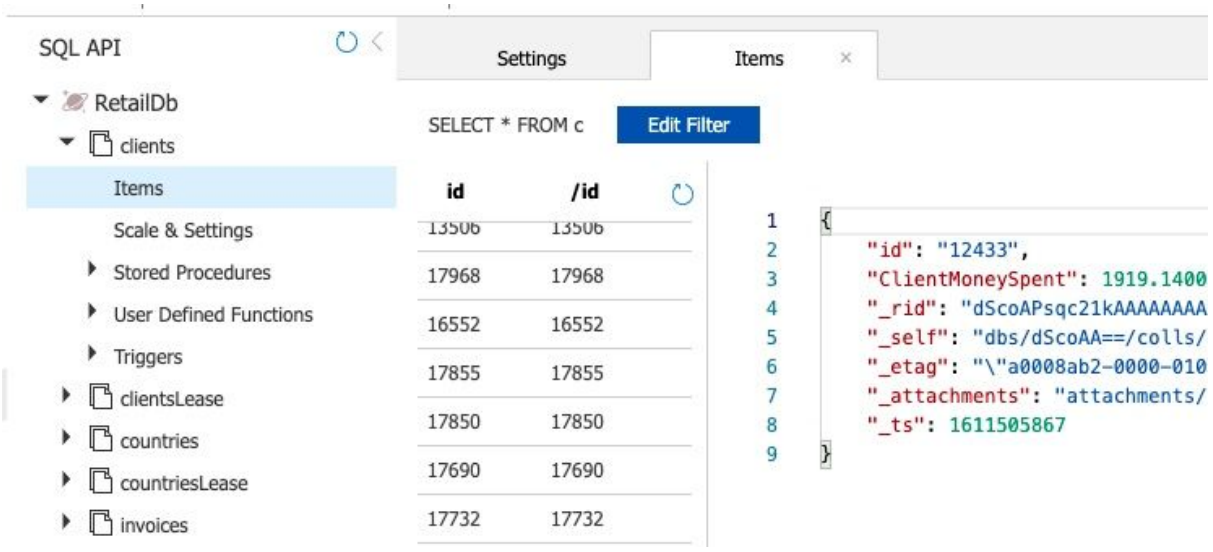
Exercises

Exercise 1

1. Run **ChangeFeedClient** with argument **--Sub clients.** Do not stop it.
   dotnet run -p AzureCosmosSparkTutorial.ChangeFeedClient --Sub clients
2. Go to the notebook, and do exercise 1.
3. After writing dataframe to db, in ChangeFeedClient output you should see sth similar:

```
---
{"id":"1d646fd1-6a76-43db-8f45-79dd93b01545","ClientMoneySpent":1526.92}
{"id":"84e37527-b9cf-4ab2-9543-829a39ba0255","ClientMoneySpent":261.28000000000003}
{"id":"4357c014-8929-4f16-8180-db771487303b","ClientMoneySpent":236.97}
{"id":"3b4a8069-81d5-471b-b636-a66b16762cd9","ClientMoneySpent":360.05000000000007}
{"id":"374a3b80-2c39-4ef2-9bf6-2ebd55303e3b","ClientMoneySpent":136.24}
{"id":"0f470b64-3e9d-4e80-b75e-9e6162ad60f3","ClientMoneySpent":100.19999999999999}
```

4. Go to cosmosDB Data Explorer, and view items in clients container, you should see sth similar:



5. Stop **ChangeFeedClient**

Exercise 2

1. Run **ChangeFeedClient** with argument **--Sub countries.** Do not stop it.
   dotnet run -p AzureCosmosSparkTutorial.ChangeFeedClient --Sub countries
2. Go to the notebook, and do exercise 2.
3. Run writeStream, and do not stop it.
4. In ChangeFeedClient, you should see records aggregated on data pushed to invoices container before (prerequisites), e.g.:

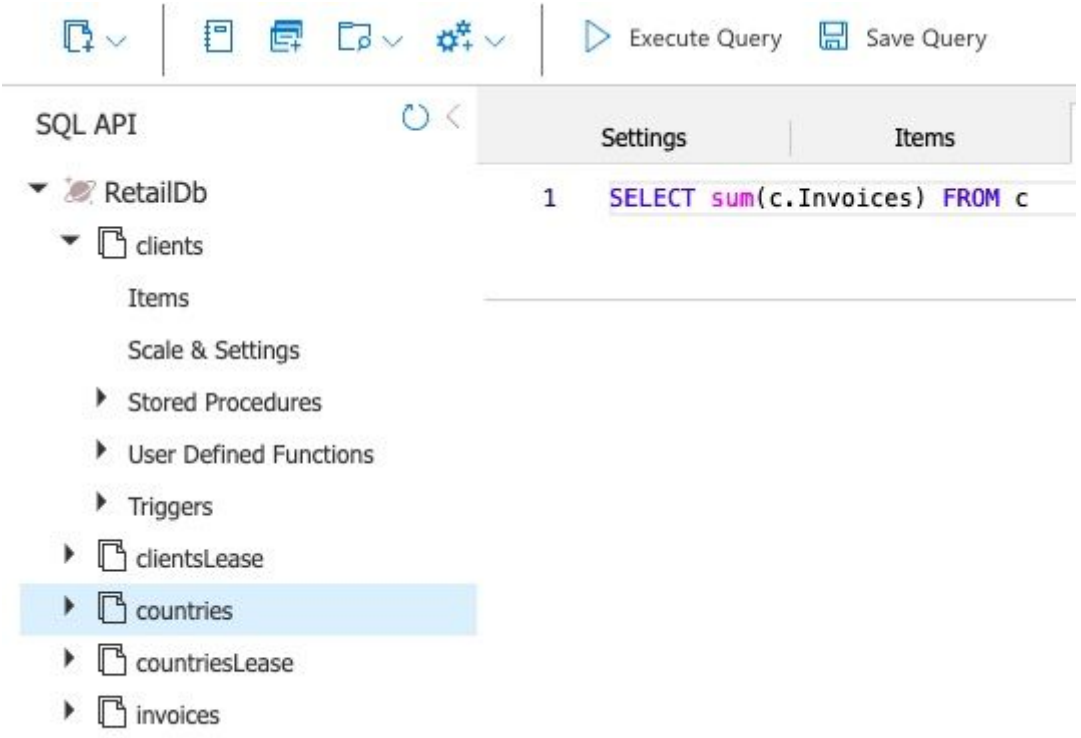
```
---
{"id":"France","Invoices":4}
{"id":"Germany","Invoices":10}
{"id":"Belgium","Invoices":1}
{"id":"Italy","Invoices":1}
{"id":"EIRE","Invoices":7}
{"id":"Lithuania","Invoices":3}
{"id":"Norway","Invoices":1}
{"id":"Spain","Invoices":1}
{"id":"Switzerland","Invoices":1}
{"id":"Poland","Invoices":1}
{"id":"Portugal","Invoices":1}
```

5. Run DataGenerator with: **--Skip 500 --Take 1500**
   The generator will push the next 1500 rows, so the total number of rows in the *invoices* container after the whole push should be equal to 2000.

6. Simultaneously you should see after few moments, that in changeFeedClient, records are being updated in small batches, e.g.

```
---
{"id":"EIRE","Invoices":26}
---
{"id":"United Kingdom","Invoices":2243}
---
{"id":"France","Invoices":27}
{"id":"Belgium","Invoices":5}
{"id":"EIRE","Invoices":27}
---
{"id":"United Kingdom","Invoices":2278}
{"id":"Netherlands","Invoices":4}
```

7. Go to CosmosDB DataExplorer, select countries -> New SQL query, and write **SELECT sum(c.Invoices) FROM c**



The result of the query after refreshing a few times should be 2000.

8. Now you can stop the stream.