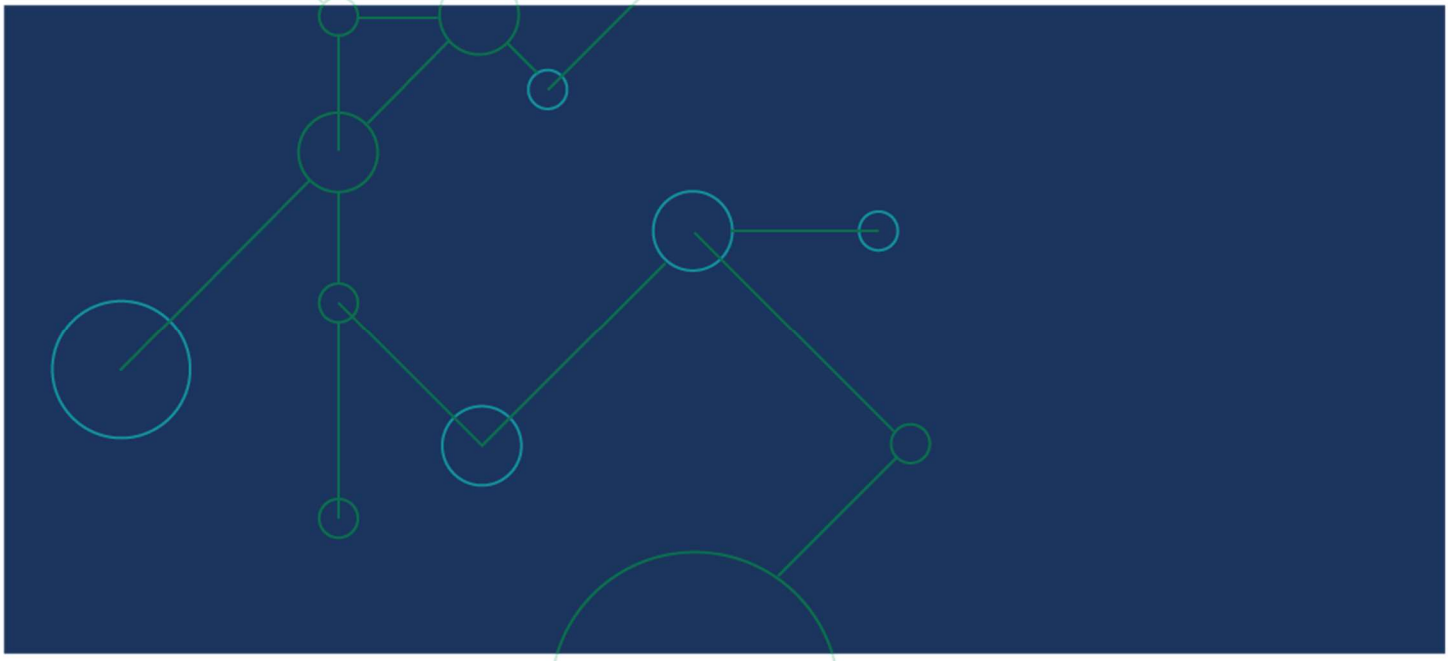
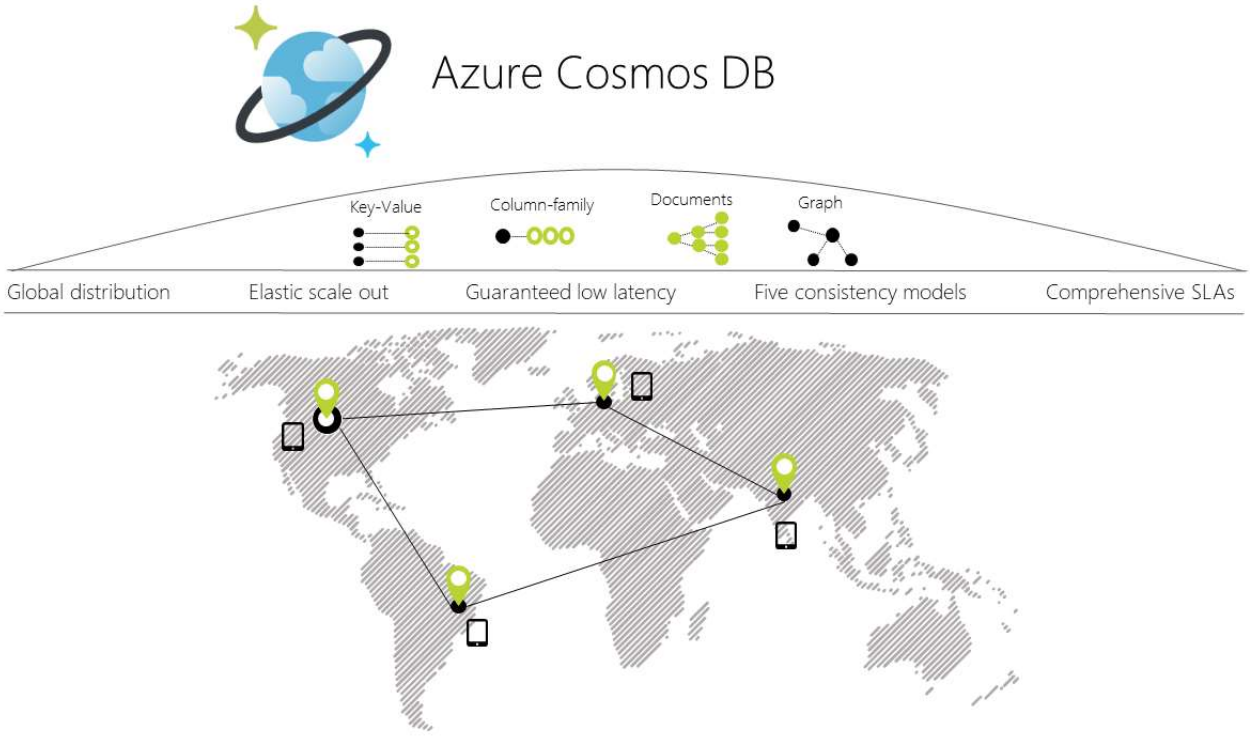
WHITE PAPER

Connecting to CosmosDB SQL API from Qlik Sense via REST Connector



Cosmos DB Overview

Cosmos DB is a fast growing, multi-modal database service in Microsoft Azure offering several API's. Currently, the SQL API is the most popular and widely used API. When you create a Cosmos DB account, you must decide which API you want to use as the data will get stored in corresponding data model/format.



Key Cosmos DB capabilities

- Turnkey global distribution
- You can distribute your data to any number of Azure regions, with the click of a button. This enables you to put your data where your users are, ensuring the lowest possible latency to your customers.
- Using Azure Cosmos DB's multi-homing APIs, the app always knows where the nearest region is and sends requests to the nearest data center. All of this is possible with no config changes. You set your write-region and as many read-regions as you want, and the rest is handled for you.
- As you add and remove regions to your Azure Cosmos DB database,

- your application does not need to be redeployed and continues to be highly available thanks to the multi-homing API capability.

- Multiple data models and popular APIs for accessing and querying data

    - The underlying atom-record-sequence (ARS) data model that Azure Cosmos DB is built on natively supports multiple data models, including document, graph, key-value, table, and column-family data models.

    - APIs for the following data models are supported with SDKs available in multiple languages:

        - SQL API: A schema-less JSON database engine with rich SQL querying capabilities
        - MongoDB API: A massively scalable *MongoDB-as-a-Service* powered by Azure Cosmos DB platform. Compatible with existing MongoDB libraries, drivers, tools, and applications
        - Cassandra API: A globally distributed Cassandra-as-a-Service powered by Azure Cosmos DB platform. Compatible with existing Apache Cassandra libraries, drivers, tools, and applications
        - Gremlin API: A fully managed, horizontally scalable graph database service that makes it easy to build and run applications that work with highly connected datasets supporting Open Gremlin APIs (based on the Apache TinkerPop specification, Apache Gremlin)
        - Table API: A key-value database service built to provide premium capabilities (for example, automatic indexing, guaranteed low latency, global distribution) to existing Azure Table storage applications without making any app changes.

The SQL API can be interacted with using ODBC, REST, or native code bases such as .NET (Core and Standard), Java, Go, Node.js, or Python.

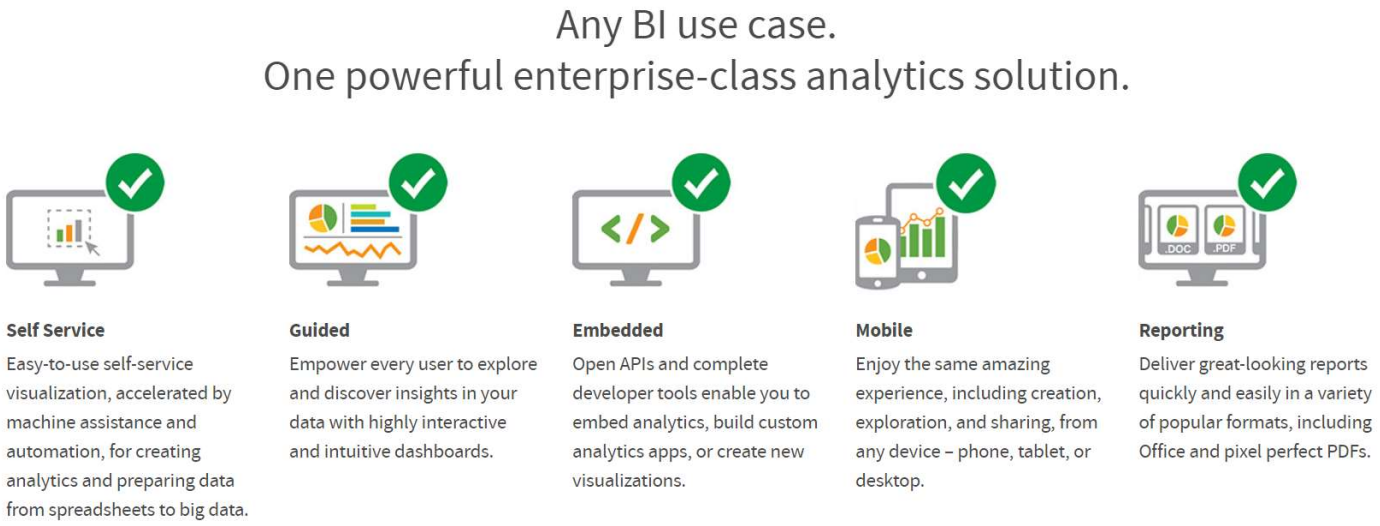
There are many connectivity methods validated with Qlik Partner Engineering:

- **SQL API** using ODBC Connector in Qlik Sense
- **SQL API** using REST Connector in Qlik Sense
- **MongoDB API** using the Qlik Sense MongoDB Connector (in Beta as of Oct 2018)
- **MongoDB** API using the gRPC connector for Qlik Core

The focus of this document is the details of connecting to the SQL API via the ODBC Connector.

About Qlik Sense

Qlik Sense gives you data superpowers. Easily combine all your data sources, no matter how large, into a single view. Qlik's Associative engine indexes every possible relationship in your data so you can gain immediate insights and explore in any direction your intuition takes you. Unlike query-based tools, there's no pre-aggregated data and predefined queries to hold you back. That means you can ask new questions and create analytics without having to build new queries or wait for the experts.



Interactive analysis, without boundaries

Ask any question and quickly explore across all your data for insight, using global search and interactive selections. All analytics update instantly with each click, no matter how deep you go, furthering analysis or pivoting your thinking in new directions. There's is no limit to exploration and no data left behind.

Simply smarter visualizations

Innovative visualizations put your data in the right context to answer any question. Explore the shape of data and pinpoint outliers. Use advanced analytics integration and geographic calculation to broaden insight. And it's fully interactive - easily pan, zoom, and make selections to find insights visually.

Create and explore on any device

Explore, create, and collaborate on any device, directly at the point of decision. Qlik Sense is built from the ground up with responsive mobile design and touch interaction. Build analytics apps once, and they'll work everywhere, on desktop, tablet, or mobile devices.

Cosmos DB Overview

Cosmos DB is a fast growing, multi-model database service in Microsoft Azure that offers several APIs. Currently, the SQL API is the most popular and widely used API. When you create a Cosmos DB Account, you must decide which API you want to use (this could change in the future where they merge and you can use all APIs under one account).
The API types are the following:

- SQL API
- MongoDB API
- Cassandra API
- Gremlin API
- Table API

The SQL API can be interacted with using REST or native code such as .Net (Core and Standard), Java, Go, Node.js, or Python.
https://docs.microsoft.com/en-us/azure/cosmos-db/introduction
There are a number of connectivity methods validated with Qlik Partner Engineering:

a) SQL API using REST Connector in Qlik Sense and QlikView
b) MongoDB API using the Qlik Sense MongoDB Connector (Beta currently, Oct 2018)
c) MongoDB API using the ODBC connector*
d) Mongo DB API using the gRPC connector for Qlik Core

The focus of this document is the details of connecting to the SQL API via the REST Connector.

Authentication

The authentication scheme in Cosmos DB using SQL API is handled via a two-step process. First you generate an authentication token with your masterKey (Account Key) and a few other header parameters and then this token will be valid for one hour and can be used in subsequent calls to get records.
The authentication token cannot be retrieved using a REST endpoint; it must be generated in code. Therefore, an Azure Function was created in C# which takes in the aforementioned parameters and returns the token and date to be used in subsequent calls to get data back from Cosmos DB. For more information on authenticating REST requests to Cosmos DB please see this article.
The Qlik REST Connector does not provide this authentication methodology in the User Interface, therefore the majority of this document is to illustrate how this is done.

Setting up the Authentication Azure Function App

Azure Functions is an event-driven serverless compute experience that works on a pay-as-you-go model, with scale on demand. It allows you to build microservices faster with a serverless architecture. This makes it a lightweight, extremely cheap way to run the code that will generate an authentication token for Cosmos DB.

Follow these steps to deploy an Azure Function in your subscription that you can use to authenticate your requests throughout the rest of this article.

1. Download the code from this Github repository: https://github.com/chrislarsenqlik/CosmosDB_QlikREST

2. Note: You only need to either the C# or the Node version. We will use C# for this example, but instructions for deploying the Node version can be found in the Github README.

3. Open PowerShell and navigate to the “CSharp_Authentication” folder inside of the downloaded code.

4. ‘cd Your\Download\Location\CSharp_AuthFunction’

5. From PowerShell, run the deploy script to deploy a new Azure Function using the command below. Add the resource group you wish to deploy to and your preferred region. The script will first ask you to log in to your Azure subscription, then it will deploy four new resources: the function app with a function already in it, an app service plan that manages your pricing, application insights that gives you telemetry on your app, and a storage account which serves as the storage for your new function.

6. ‘.\deploy.ps1 run.csx template.json <Resource group name> <Your preferred Azure region>’

7. Once you have deployed your function, get the endpoint you can use to generate your authentication token. There are two ways to find this endpoint.

8. Option 1: Recreate it yourself from the PowerShell output

9. The endpoint is in this format: https://fn<uniqueResourceNameSuffix>.azurewebsites.net/api/MyFunction

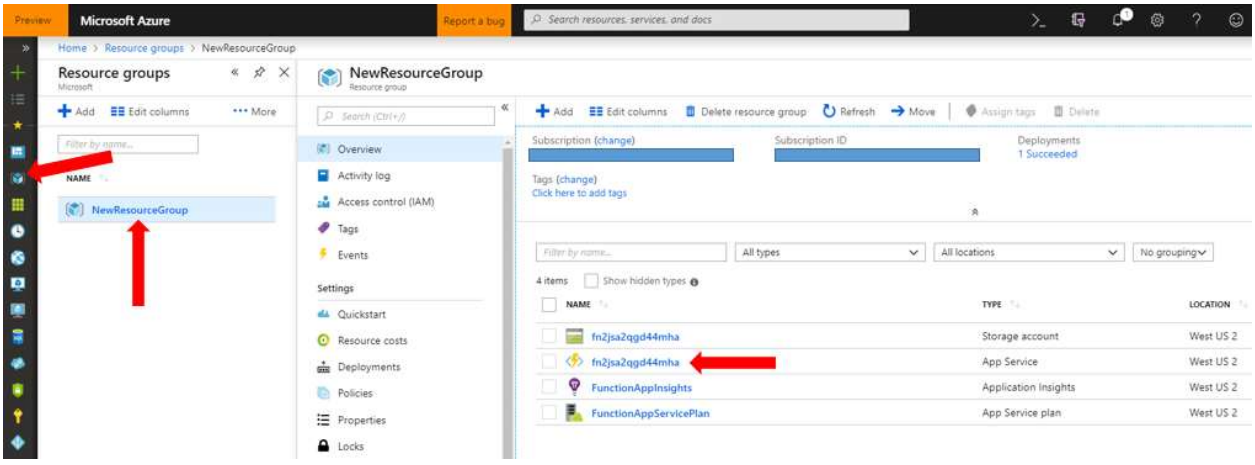
10. The endpoint for the sample PowerShell output below would be: https://fn2jsa2qgd44mha.azurewebsites.net/api/MyFunction

11.
```
DeploymentName          : template
CorrelationId           : 754cbb65-ac82-426b-9744-ce6d4db6dfc1
ResourceGroupName       : NewResourceGroup
ProvisioningState       : Succeeded
Timestamp               : 10/5/2018 8:16:27 PM
Mode                    : Incremental
TemplateLink            :
TemplateLinkString      :
DeploymentDebugLogLevel :
Parameters              : {[uniqueResourceNameSuffix, Microsoft.Azure.Commands.ResourceManager.Cmdlets.SdkModels.Deploymen
tVariable], [functionFile, Microsoft.Azure.Commands.ResourceManager.Cmdlets.SdkModels.DeploymentVariable],
                          [applicationInsightsLocation, Microsoft.Azure.Commands.ResourceManager.Cmdlets.SdkModels.Deploym
entVariable]}
ParametersString        :

                          Name                 Type                       Value
                          ===============      =========================  ==========
                          uniqueResourceNameSuffix  String                2jsa2qgd44mha
                          functionFile         String                     #r "Newtonsoft.Json"

                          using System.Net;
                          using Newtonsoft.Json;
                          using Microsoft.AspNetCore.Mvc;
                          using Microsoft.Extensions.Primitives;

                          public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
                          {
                              //JSON input example
                              //{
                              // "verb": "GET",
                              // "resourceType": "docs",
                              // "resourceId": "dbs/video_gamesdb/colls/vgdata",
                              // "masterKey": "MASTER HERE"
                              //}
```

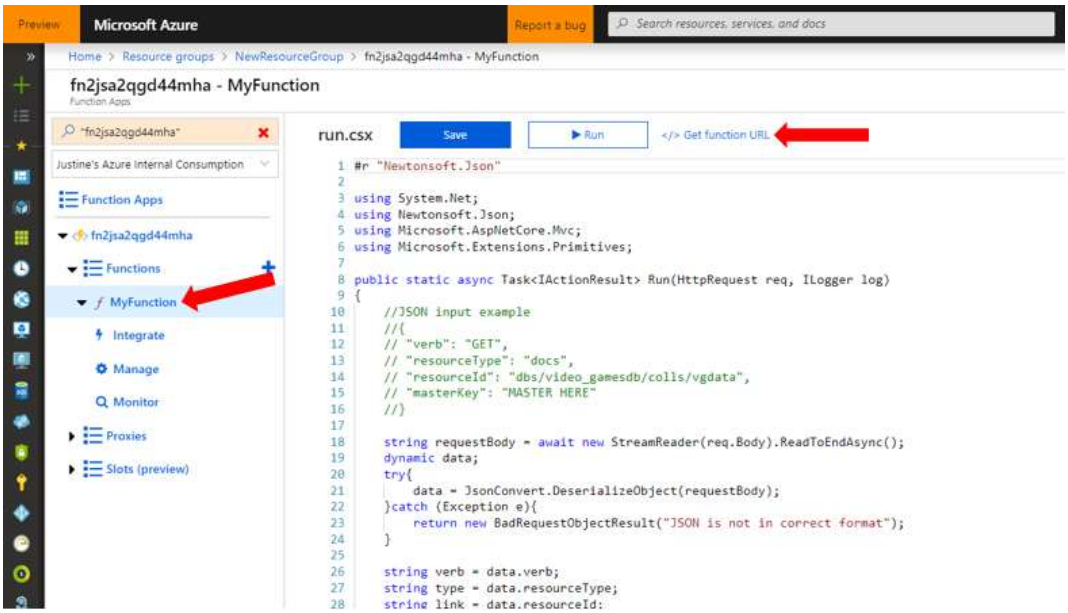
12. Option 2: Find the endpoint from the Azure Portal

13. Once the script has finished running, sign in to the Azure Portal at https://portal.azure.com

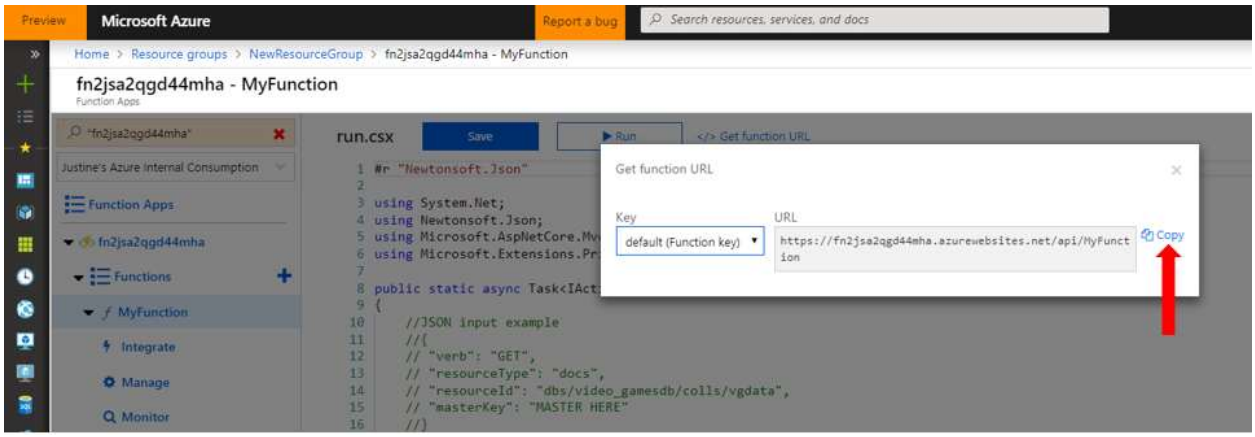
14. From the side panel of options, select the blue cube to view your resource groups. Then select the new resource group you created your function in, and select the Azure Function App.

15.


16. Select “MyFunction” and then click the “</> Get function URL” link to get the endpoint for your function.

17. 

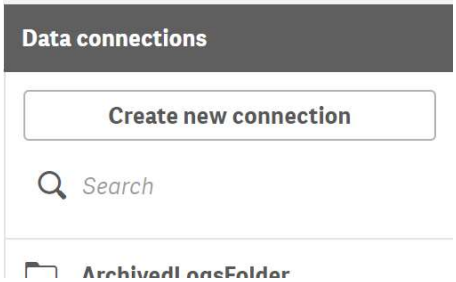
18. This is the endpoint you will use to generate authentication tokens for Cosmos DB.



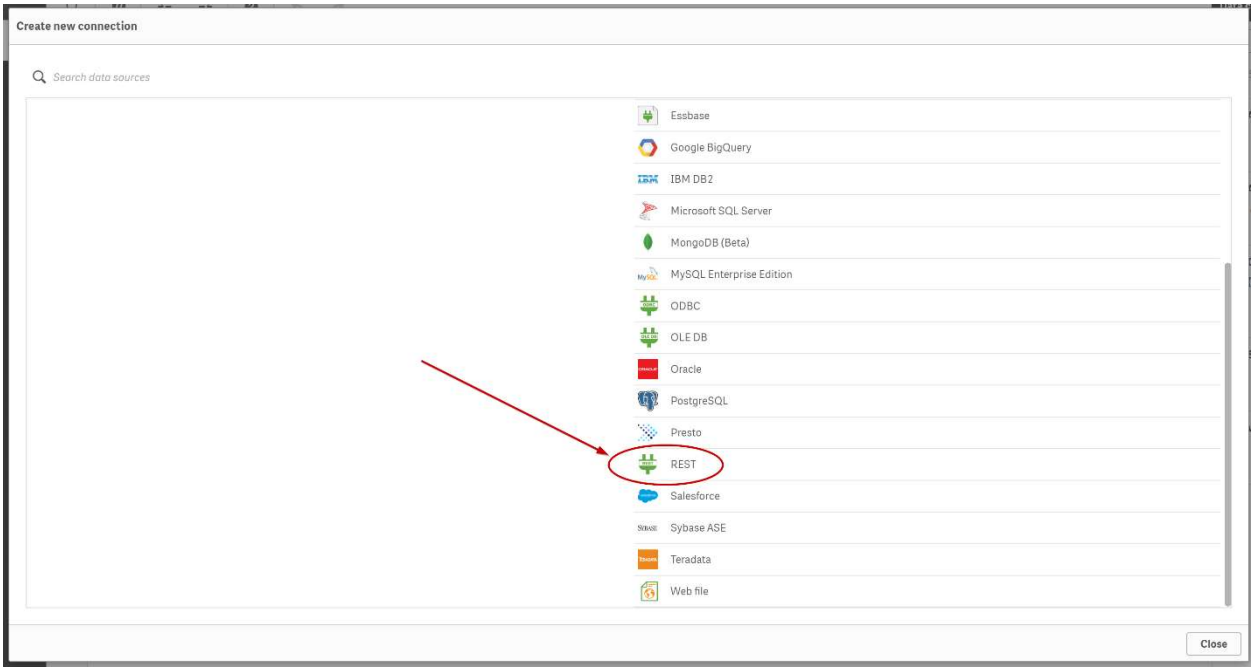
Creating the Authentication REST Connection

Creating the Authentication REST Connection in Qlik Sense to the Azure Function App is done in the following manner:

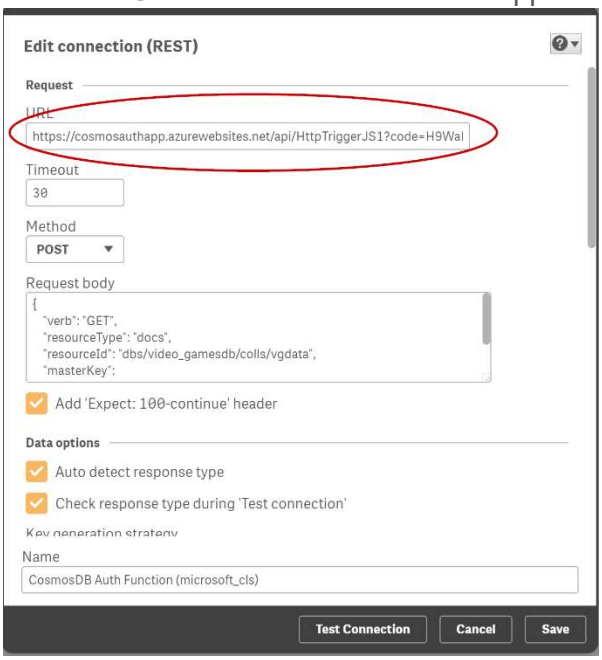
1. In Qlik Sense, go to the Data Load Editor and choose Create new connection



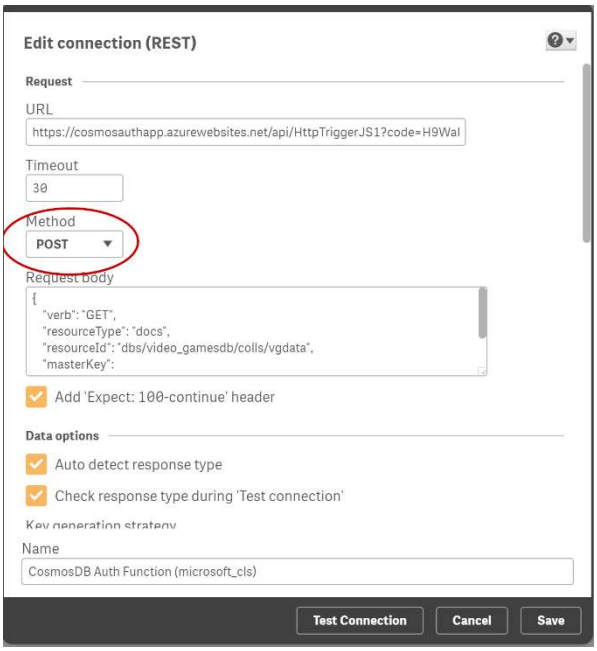
2. Choose the REST Connector



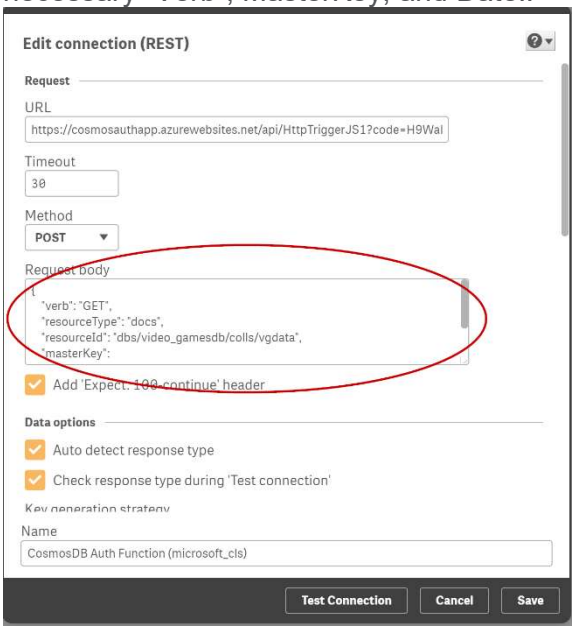
3. Enter the URL for the Azure Function App that was deployed:



4. Choose POST for the Method



5. For the Request Body, enter the parameters for the collection you will want to connect to, necessary “Verb”, MasterKey, and Date..

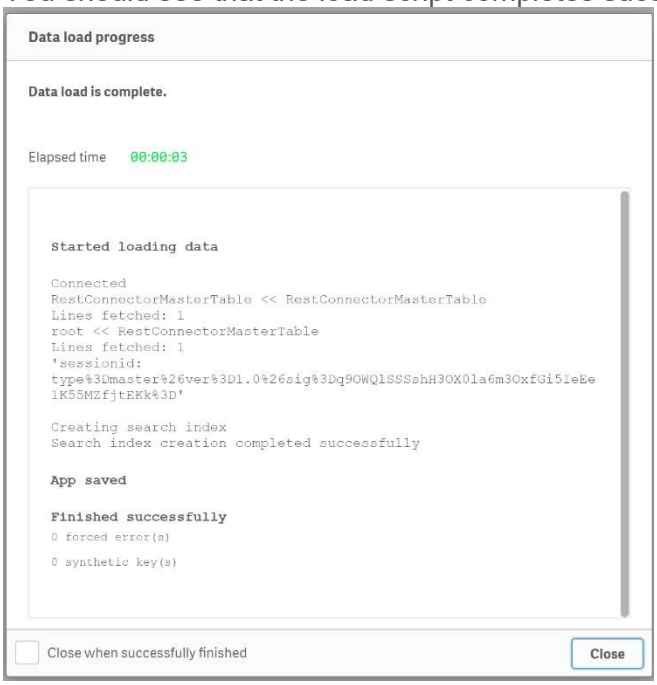


Here is the format of the JSON required in the Request Body

```
{
   "verb": "GET",
   "resourceType": "docs",
   "resourceId": "dbs/video_gamesdb/colls/vgdata",
   "masterKey":
"BlahBlahBlahBlahBlahMh0yb39aaApzk5t8p8KTmWSLmetXFqjDNNNMO4iXTaV2KMGWM
wPj8LvQ2HlBtAg0Lxhog=="
}
```

NOTE: It’s good practice to test this in Postman or another REST client before attempting to run in Qlik Sense, it should work outside of Qlik first.

6. Give it a name and click “Test Connection”. If it succeeds, click Create and it will insert the script into the load script. Click Load Data.
7. You should see that the load script completes successfully, similar to this:

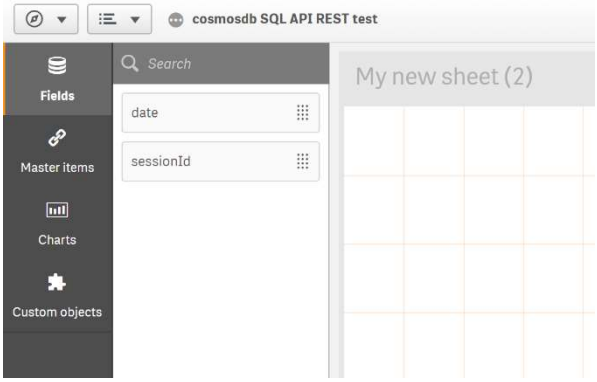


8. Note there is an extra piece of text in this above output, it’s the sessionId traced out. You should edit the injected script and append this scriptlet to store the sessionId (authorization token) and also do some basic error handling:

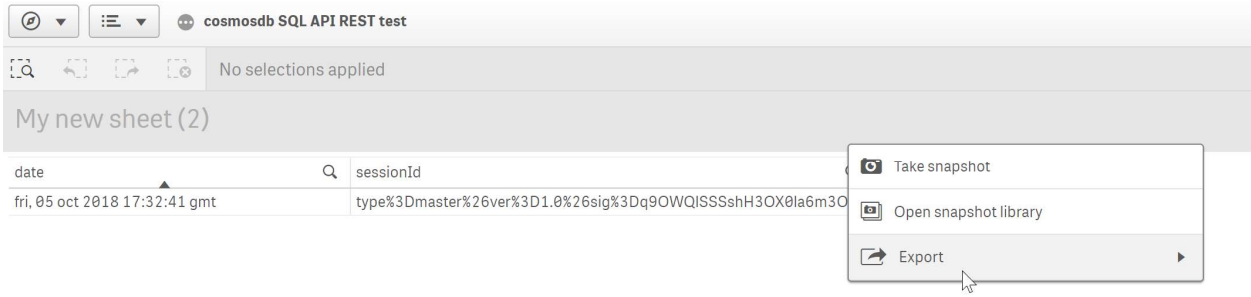
```
Let vSessionId_tmp=peek('sessionId');
set vSessionId=$(vSessionId_tmp);

Let vMS_Date_tmp=peek('date');
set vMS_Date=$(vMS_Date_tmp);

Let vMS_Version='2017-02-22';
trace 'sessionid: $(vSessionId)';

IF '$(vSessionId)'='' then
  trace AUTHORIZATION FAILED;
  exit script;
ENDIF
```

9. Load again and ensure it works, then we will move on to the next step of creating the pull of documents.
10. In order to get the “Authorization” parameter and “x-ms-date” parameter, add a new Sheet in Qlik Sense, add the fields, and export to Excel
    a. Click Edit in the new Sheet

b. On the left side, find the Fields “sessionId” and “date”



c. Drag date onto the canvas, it will create you a Table object by default, then drag sessionId as well
d. Click Done and you will see the table with the 2 columns/values.
e. Right click on the table object and choose Export

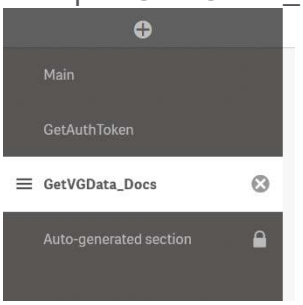


f. Then choose Export Data and it will export an Excel file, open it. We’re ready to move on to the next step which is to create the Connection for loading the document records.

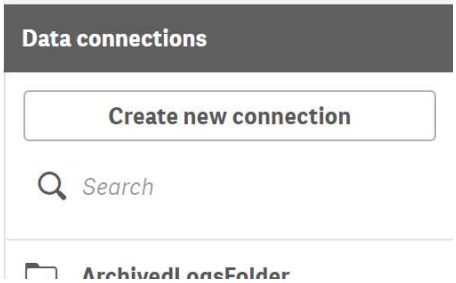
Creating the Documents REST Connection

Creating the REST Connection in Qlik Sense to get the Documents in the CosmosDB document collection is done in the following manner:

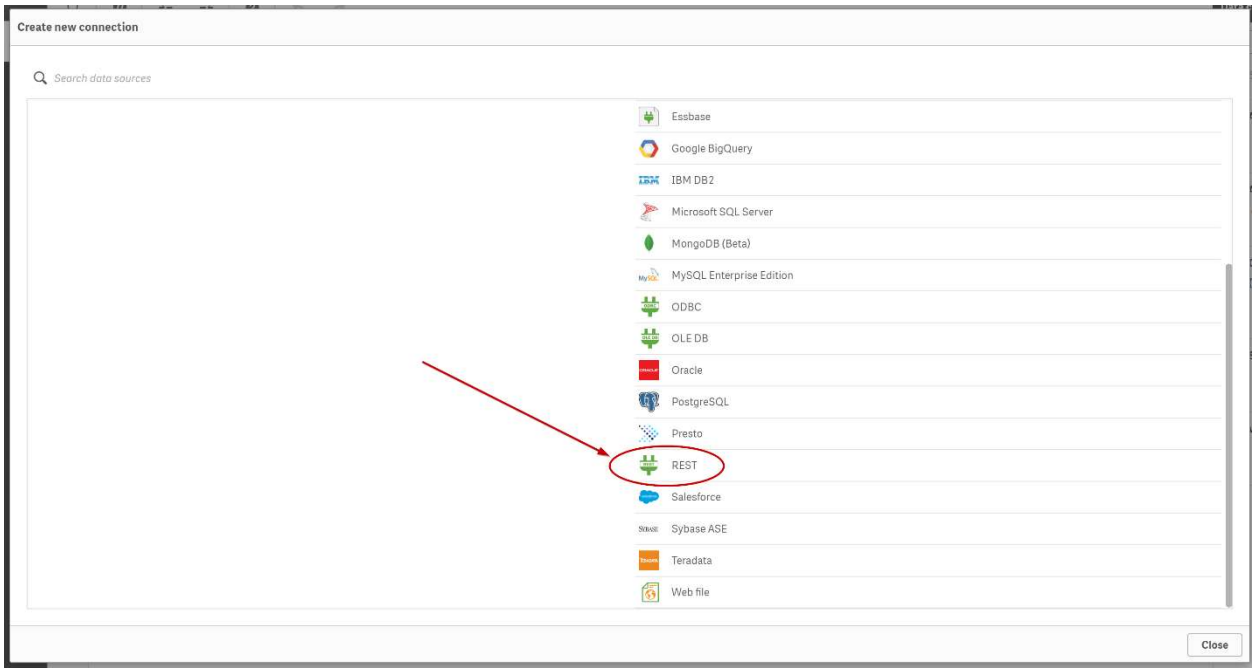
g. Go back to the Load Script in Qlik Sense and add another section of script, in this example “GetVGData_Docs”



h. …and choose Create new connection

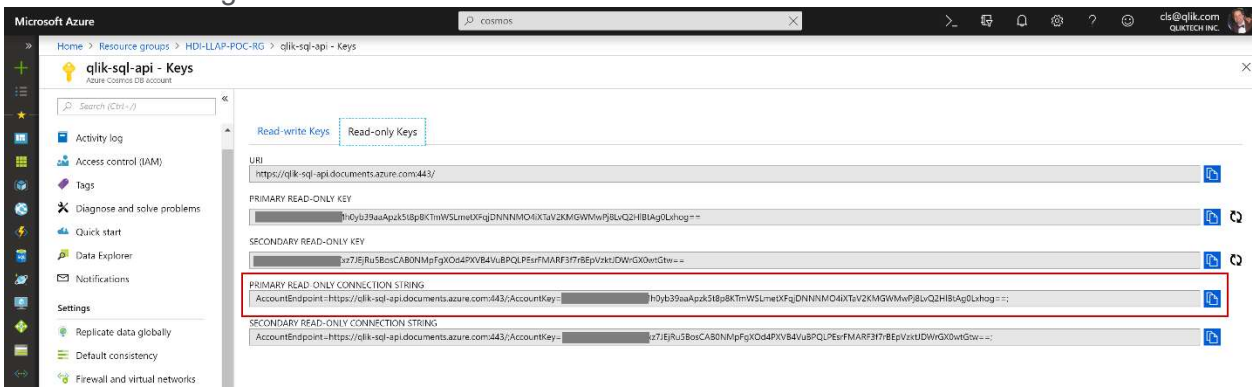


i. Choose the REST Connector

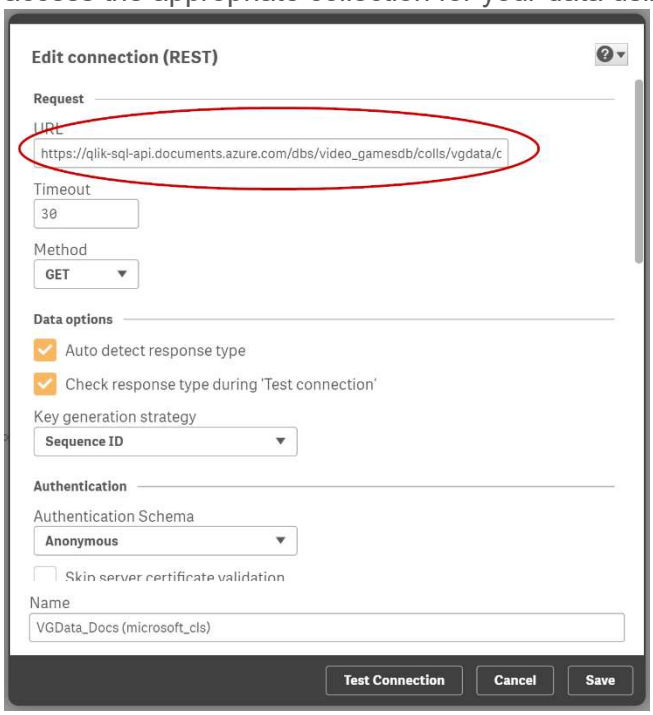


For the masterKey and Base URL, get that from the Azure Portal, go to the CosmosDB Account for the SQL API created, and copy the Primary Read-Only

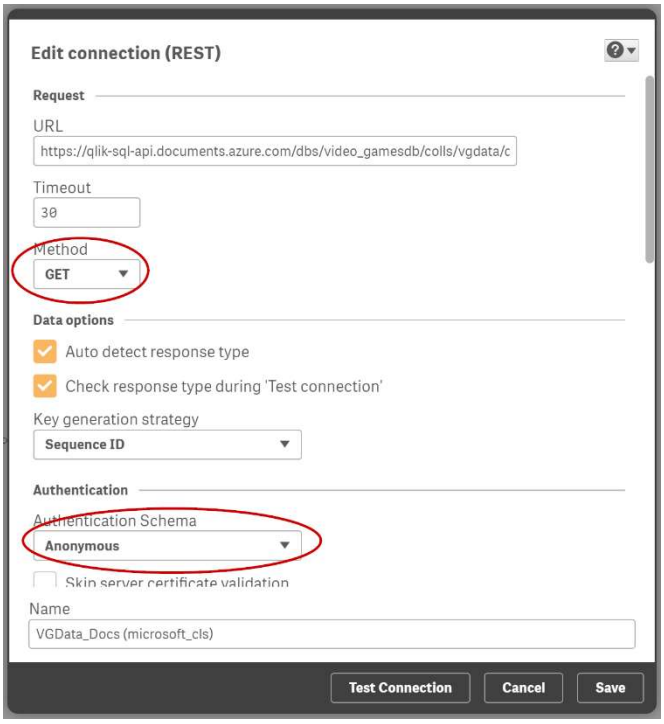
Connection String URL:



j. Construct the URI with the DNS name of the connection string from Azure portal, and access the appropriate collection for your data using the following format:



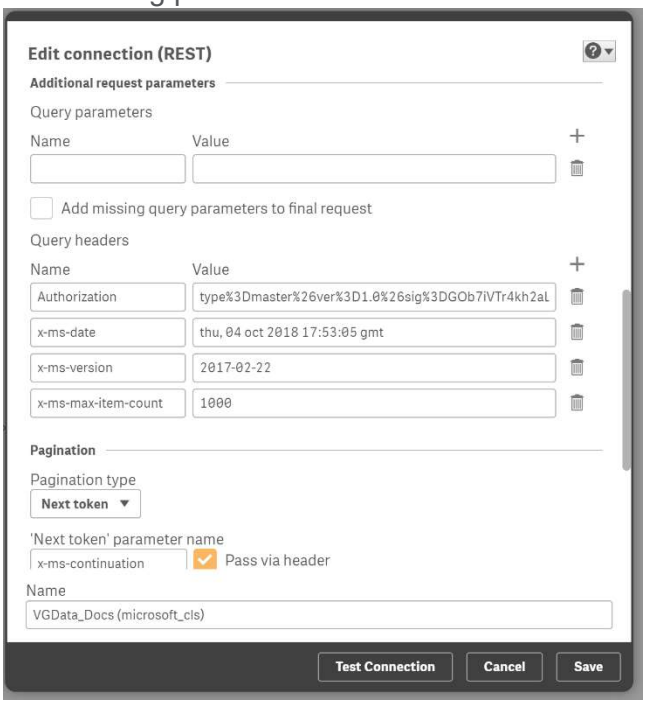
k. The method is “GET” and Authentication Schema is Anonymous



The first time we create the connection to the document collection (synonymous with a Table in a database), the appropriate header parameters need to be hardcoded. In the later steps we will variablize these to make it dynamic, but on first connection we need to hardcode them.

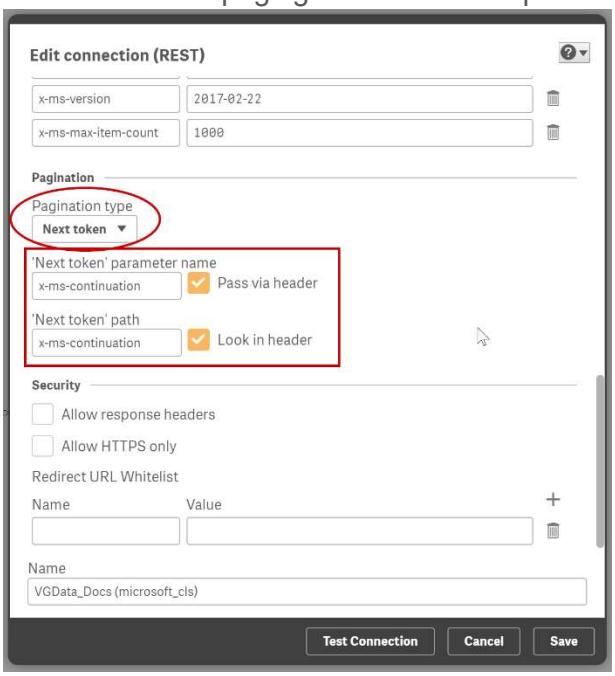
l. Open the Excel file you downloaded from the previous Authentication step load

m. Scroll down in the REST Connector dialog until you get to “Query headers” and add the following parameters:



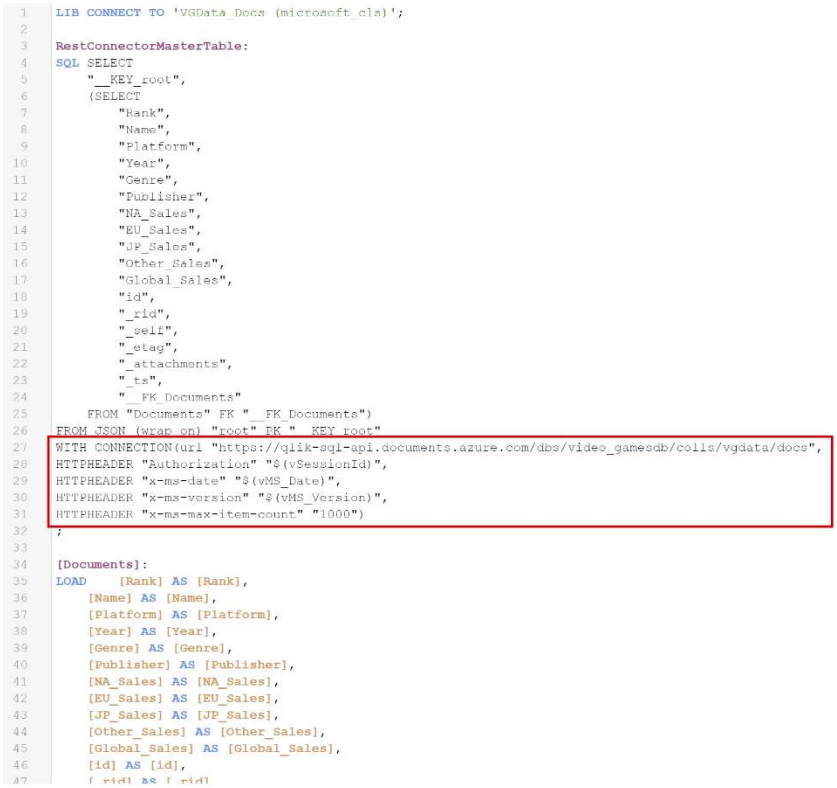
n. Authorization is the sessionID from the Excel file downloaded. Copy and paste it in. (NOTE: An extra line break could be added, it comes in from Excel)

o. Do the same with "`x-ms-date`" using the "`date`" column in the Excel document
p. Then add "x-ms-version", this should generally be the latest.. you can get the versions from here: https://docs.microsoft.com/en-us/rest/api/cosmos-db/#supported-rest-api-versions
q. "`x-ms-max-item-count`" is used for paging, this is the maximum amount of records (documents) to receive per page.
r. Paging is also necessary, and the way this works in CosmosDB SQL API is by using the "NextToken" paging method. Set it up like this:



As it is the first time you are creating it, give it a name, and hit Create.
The script will be injected into the Load Script

s. In order to make it dynamic, we add the WITH CONNECTION() syntax under the FROM clause:

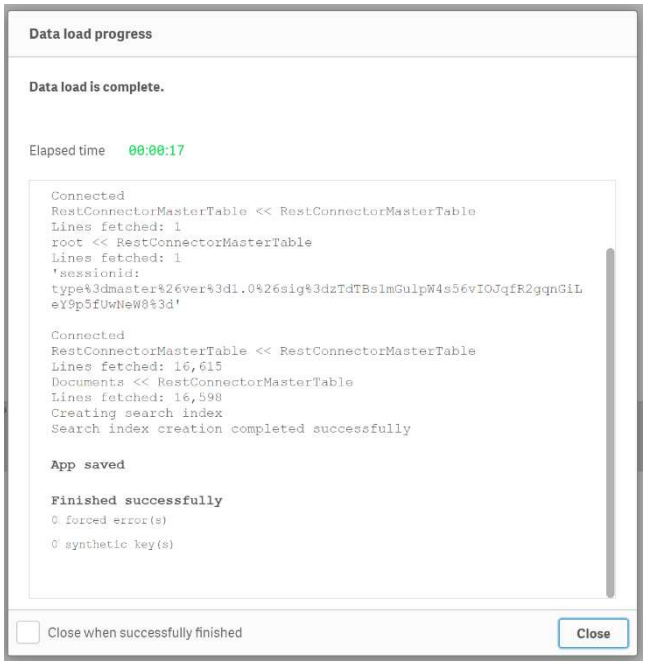


```
LIB CONNECT TO 'VGData_Docs (microsoft_cls)';

RestConnectorMasterTable:
SQL SELECT
    "__KEY_root",
    (SELECT
        "Rank",
        "Name",
        "Platform",
        "Year",
        "Genre",
        "Publisher",
        "NA_Sales",
        "EU_Sales",
        "JP_Sales",
        "Other_Sales",
        "Global_Sales",
        "id",
        "_rid",
        "_self",
        "_etag",
        "_attachments",
        "_ts",
        "__FK_Documents"
    FROM "Documents" FK "__FK_Documents")
FROM JSON (wrap on) "root" PK "__KEY_root"
WITH CONNECTION(url "https://qlik-sql-api.documents.azure.com/dbs/video_gamesdb/colls/vgdata/docs",
HTTPHEADER "Authorization" "$(vSessionId)",
HTTPHEADER "x-ms-date" "$(vMS_Date)",
HTTPHEADER "x-ms-version" "$(vMS_Version)",
HTTPHEADER "x-ms-max-item-count" "1000")
;

[Documents]:
LOAD    [Rank] AS [Rank],
    [Name] AS [Name],
    [Platform] AS [Platform],
    [Year] AS [Year],
    [Genre] AS [Genre],
    [Publisher] AS [Publisher],
    [NA_Sales] AS [NA_Sales],
    [EU_Sales] AS [EU_Sales],
    [JP_Sales] AS [JP_Sales],
    [Other_Sales] AS [Other_Sales],
    [Global_Sales] AS [Global_Sales],
    [id] AS [id],
    [_rid] AS [_rid]
```

t. For quick copy/paste, here is the text:

WITH CONNECTION(url "https://qlik-sql-api.documents.azure.com/dbs/video_gamesdb/colls/vgdata/docs",
HTTPHEADER "Authorization" "$(vSessionId)",
HTTPHEADER "x-ms-date" "$(vMS_Date)",
HTTPHEADER "x-ms-version" "$(vMS_Version)",
HTTPHEADER "x-ms-max-item-count" "1000")

a. Click Load Data, and see if it loads. If it's successful, it should look like this:



Yippeeee! Now it's ready for deployment.

If you need to implement additional document collections do the same step as above. Just remember this is how you create a new connection to a document collection:

1. The Authorization headers need to be hardcoded the first time, and then the connection can be created, and then
2. The text with "WITH CONNECTION()" and the variables are added under the FROM clause