Cosmos DB

Lab 10

by

Olena Bolila, Nishava Inc.

Deep Azure @McKesson

Lab's Goals

1. Familiarize with Document DB;
2. Migrate data using Document DB Migration Tool;
3. Build web app with .NET;
4. Show ease of working with Cosmos DB via Python;
5. Launch Cosmos DB instances via bash scripts;
6. Introduce to serverless architecture using CosmosDB and Azure Functions.

My Working Environment and Tools

-Windows 10 Home
-VS 2017 .NET
-Anaconda’s python 3.4
-DocumentDB Migration Tool

Why Cosmos DB?

It allows a company of any size to establish a powerful global presence quickly and easily without a major initial expenditure.

--What is Cosmos DB?

-It is a turnkey globally distributed, multi-model database system sold under the SaaS model. Cost is determined by data throughput and used storage capacity.

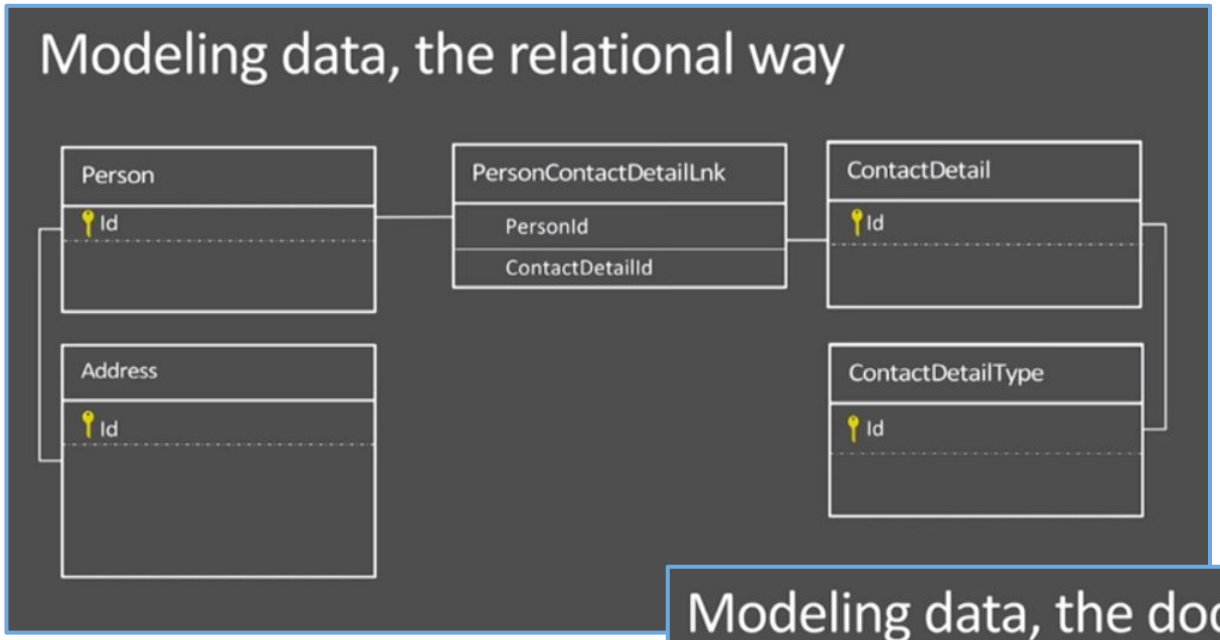
With Cosmos DB, Microsoft wants to build one database to rule them all

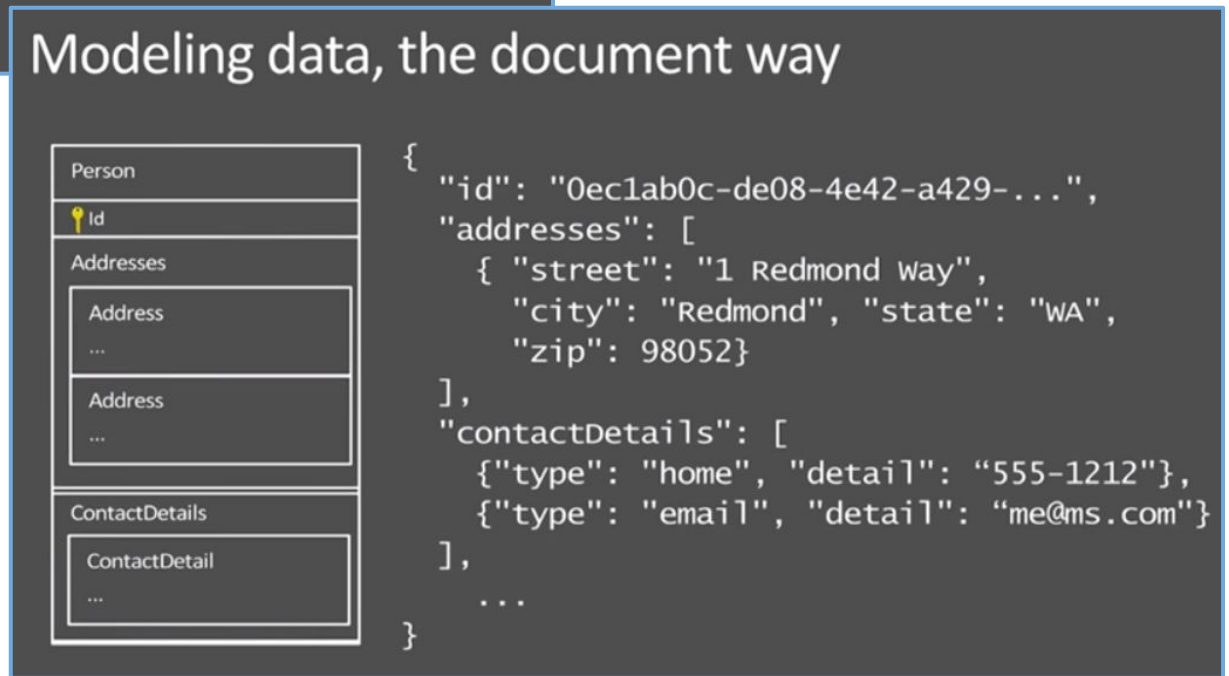
So while you could also use the MongoDB APIs to access your data in DocumentDB, Cosmos DB also features support for SQL, Gremlin and Azure Tables — and the team plans to launch a large number of similar driver and translation layers in the near future.

"No data is born relational," Shukla told me. "In the real world, nobody thinks in terms of schemas — they think graphs or maybe JSON document if you're an IOT device. [...] We want to make sure that the systems we build have a common engine to efficiently map different data models."

https://techcrunch.com/2017/05/10/with-cosmos-db-microsoft-wants-to-build-one-database-to-rule-them-all

Why Cosmos DB?





Why Cosmos DB?

Relational databases disadvantages:

- Often LOTS of tables
- Schema-centric and schema-focused
  - **Not* always data-centric*
- Application tendencies:
  - *Complex queries*
  - *Hard to scale horizontally*
  - *Limits to scaling vertically*
  - *Tied to ORM*
  - *Multi-table joins, locking hints, configurable transaction isolation levels, "user-defined columns"*
  - *&*%#$@!($*#$!!?!?*

Why Cosmos DB?

Relational databases

- Row-oriented
- Pre-ordained schema is primary
  - *Even to the detriment of accessing applications!*
- Manual index definitions
  - *Frequent source of perf issues*
- (Mostly) non-transparent data partitioning
  - *Frequent source of scalability issues*
- Supports configurable, *local* consistency

DocumentDB

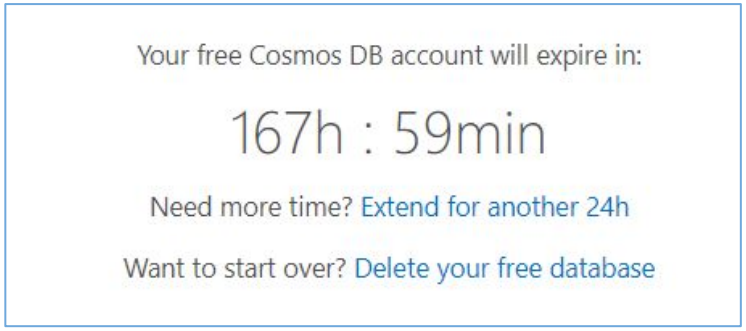
- Document-based
- Focus is the data itself
  - *Not a pre-ordained schema*
- Auto-indexing of all content
- (Mostly) transparent data partitioning
- Supports configurable, *distributed* consistency

- No server provisioning, configuration or management
  - *Focus is on the data, not the servers*
- 99.99% SLA
- <10ms reads and <15ms writes for 99% of queries
- Transparent, declarative geo-replication
  - *Failover, regional client reads, etc.*

Cosmos DB

<u>If you do not have Cosmos DB service with your subscription,</u>

- you can try out Azure Cosmos DB for free without an Azure subscription, free of charge and commitments. Click this link for Azure to create Cosmos DB instance you can work on for 168 hours for free: https://azure.microsoft.com/en-us/try/cosmosdb/



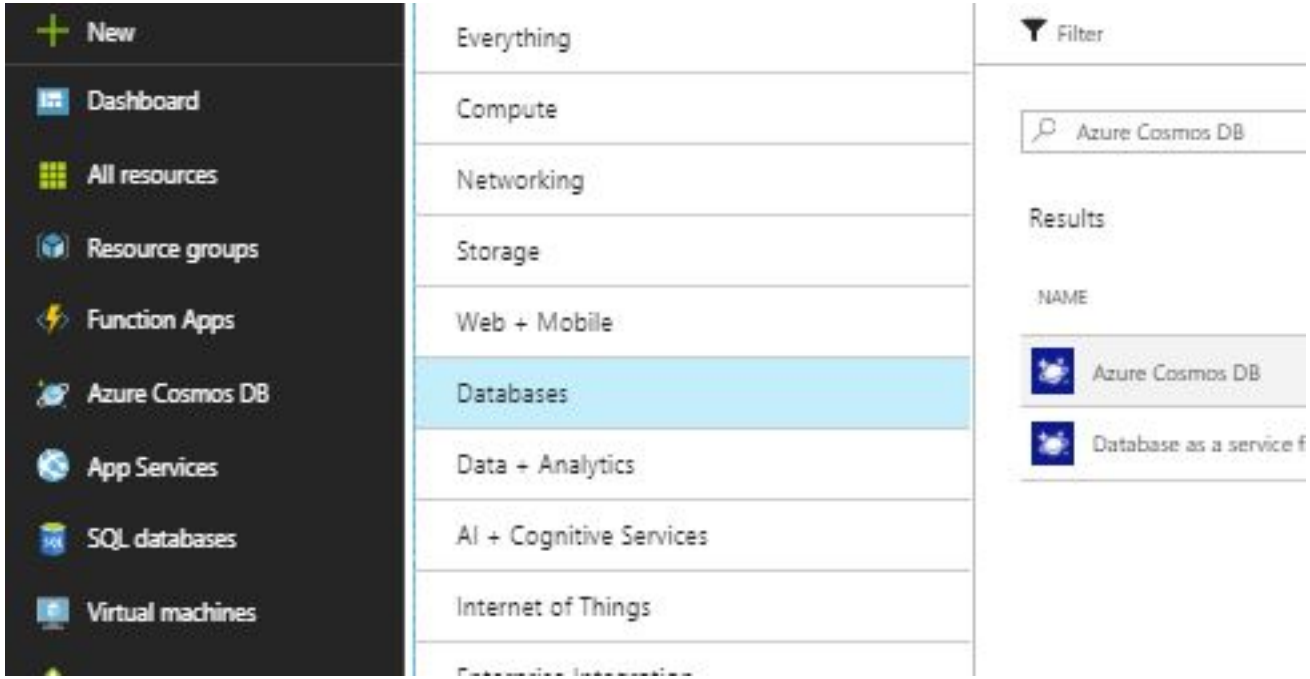
- or you can try using the Azure Cosmos DB Emulator (https://docs.microsoft.com/en-us/azure/cosmos-db/local-emulator) with a URI of https://localhost:8081 and a key of

```
C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XIw/Jw==
```

Create Cosmos DB account via Portal

<u>Assuming you did not create Cosmos DB as shown in the previous slide, let's instantiate it from scratch ourselves:</u>

Find Azure Cosmos DB Service in the portal left panel and click on **+Add**. Or click on +New, type "databases", type "cosmos", select Cosmos DB, Create.



Create Cosmos DB account via Portal

<u>Assuming you did not create Cosmos DB as shown in the previous slide, let's instantiate it from scratch ourselves:</u>

Provide some ID: "lenadocumentdbdemo", pick API, resource group, Create.

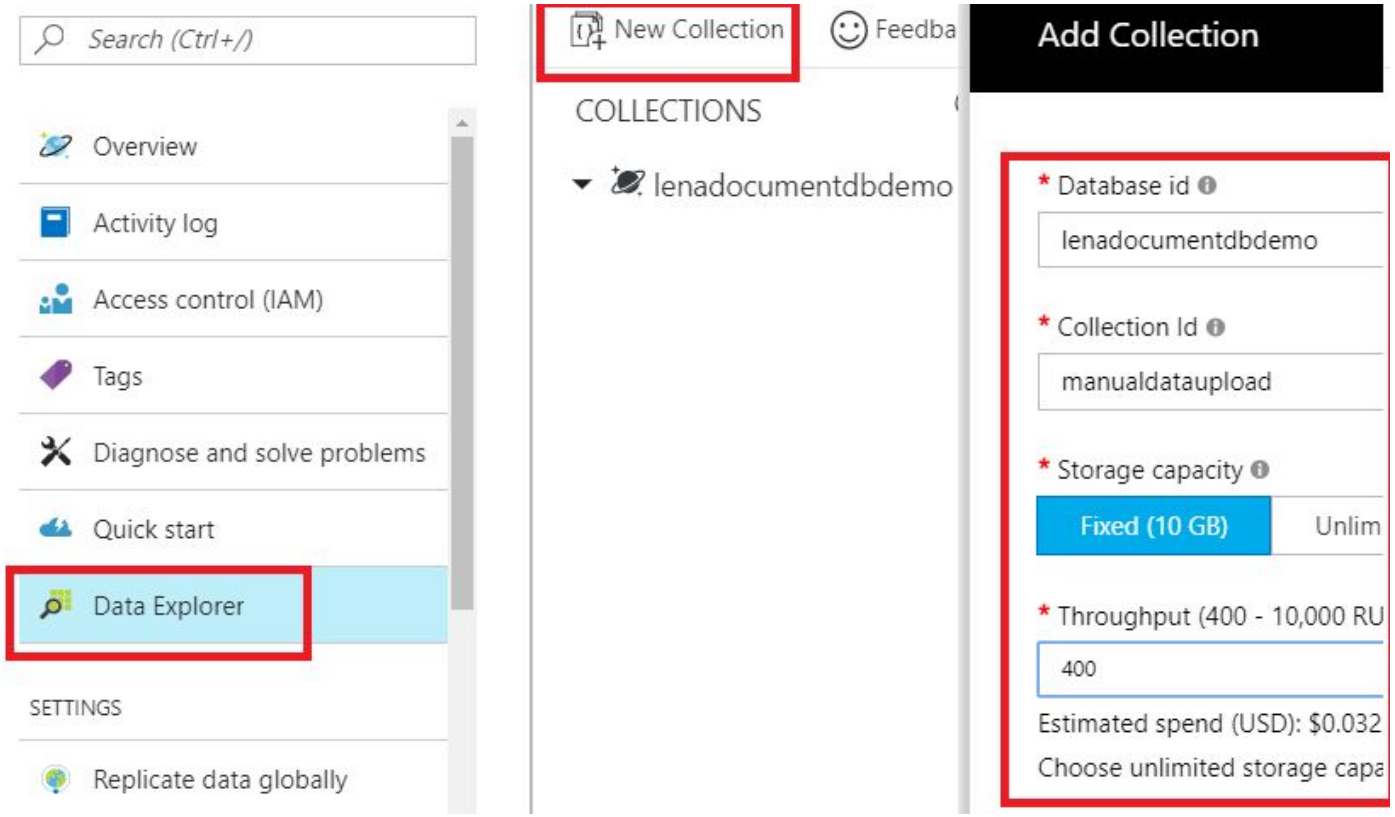
<u>API:</u> let's us choose how to interact with DocumentDB. Let's select SQL syntax.

If you click on "Document Explorer," you will see no documents found. Let's add some data.

Create a database and a collection in it

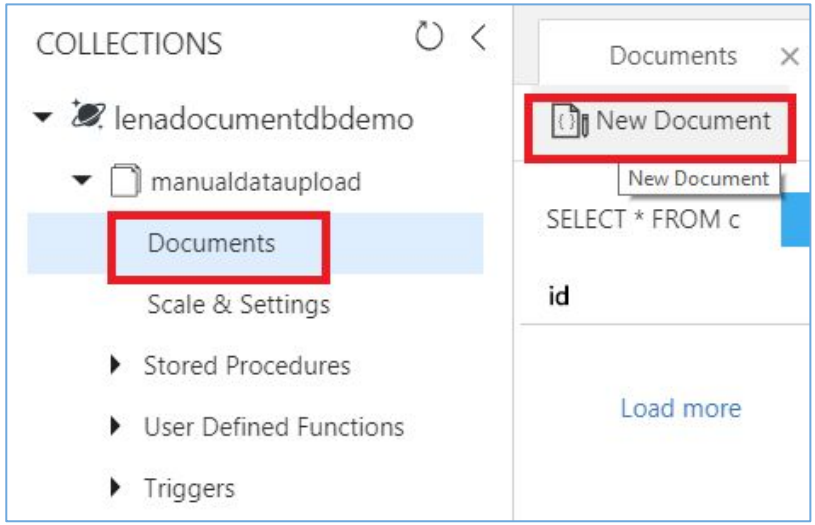
*In DocumentDB each database is a container for one or more collections and each collection can hold up to 10 GB of schema free JSON documents.

Data Explorer, New Collection, add collection information (name the database; name collection; for storage capacity, select fixed 10 GB; for throughput, select minimum: 400; leave partition key blank). OK.



Manually add sample data to your collection

Go to Cosmos DB service, my DB instance, Data Explorer, Collection named “manualdataupload”, Documents, New Document.



Delete contents, add below json document instead, then click Save icon on top.

```
{
   "id": "1",
   "category": "personal",
   "name": "groceries",
   "description": "Pick up apples and strawberries.",
   "isComplete": false
}
```

Manually add data to your collection

Add another document, then query either by going to **Data Explorer** (select collection of interest, New SQL Query) or by clicking on **Query Explorer.**

```
{
   "id": "2",
   "category": "shared",
   "name": "cleaning supplies",
   "description": "Buy Drano and dish detergent.",
   "isComplete": false
}
```

Run a sample query to return documents by timestamp: `SELECT * FROM c ORDER BY c._ts DESC`

You can also use Data Explorer to

-create stored procedures,

-UDFs,

-triggers to perform server-side business logic as well as scale throughput.

Add data to your collection via Document DB migration tool

Download the tool

*Pre-requisites: Microsoft .NET Framework 4.51 or higher

- Scroll down and click on red Download icon:
  https://www.microsoft.com/en-us/download/details.aspx?id=46436

- Once downloaded, unzip dt-1.7.zip to some folder of your choice (I've created 'software' folder on C drive, with 'documentDBMigrationTool' folder in it, copied zip to here, extracted everything in this directory, deleted the zip.)

- Then ran **Dtui.exe**, which is the graphical interface version of the tool (for command-line version of the tool, run **Dt.exe** instead).

Add data to your collection via Document DB migration tool

Download the tool

It imports data to Azure Cosmos DB from:

- JSON files
- MongoDB
- SQL Server
- CSV files
- Azure Table storage
- Amazon DynamoDB
- HBase
- Azure Cosmos DB collections

Add data to your collection via Document DB migration tool

Upload json records to a Cosmos DB collection

- In your Cosmos DB, create a new collection, “migrationtoolupload” for example.

- Download attached json which is a dummy data I’ve obtained from https://api.github.com/gists

- In Migration Tool GUI,
  - in Source Information:

    Import from: json files, add files, select location of your .json.
  - in Target Information:

    Export to: documentDB sequential, provide connection string. Its format is:

```
AccountEndpoint=<URI, which is CosmosDB Endpoint>;AccountKey=<Primary key which is this db’s key>;Database=<name of my database which is lenadocumentdbdemo in this case>;
```

Add data to your collection via Document DB migration tool

Upload json records to a Cosmos DB collection

- Get connection string from the Portal:

  Select your Cosmos DB instance, ‘lenadocumentdbdemo’ in my case. In settings, under ‘Keys’, copy ‘Primary Connection String’ and then add **Database** to the end.

  PRIMARY CONNECTION STRING

  AccountEndpoint=https://lenadocumentdbdem

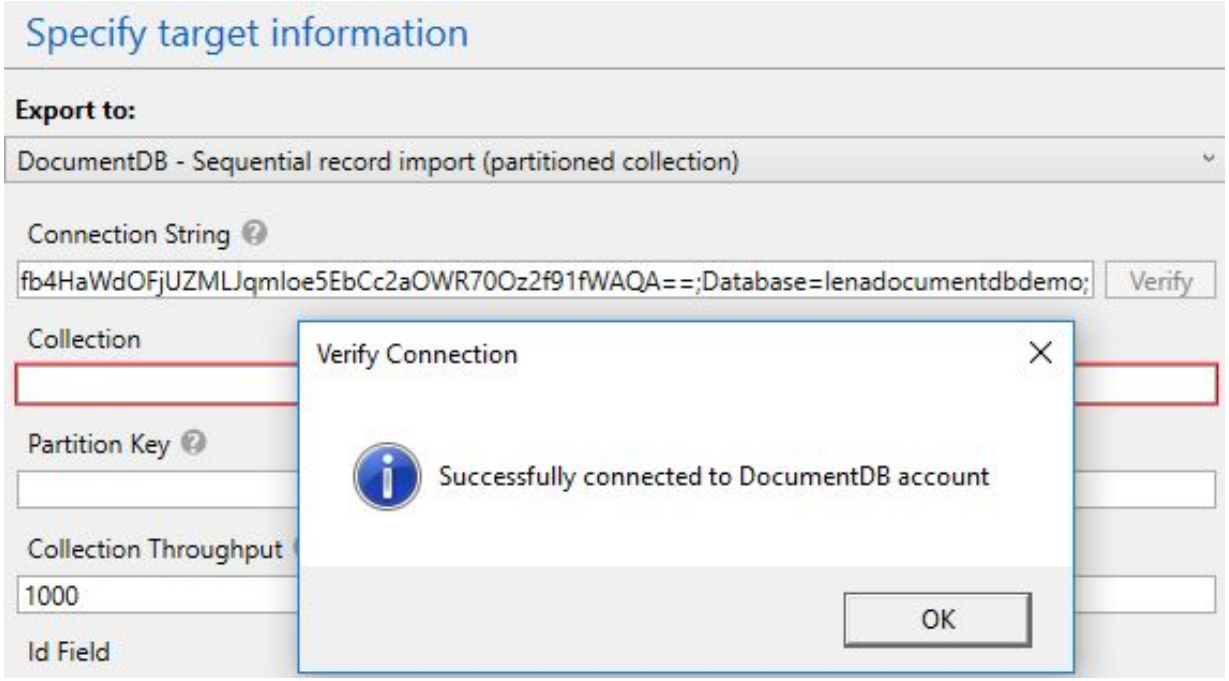
  Example: My connection string to add to ‘Connection String’ field under Target Information in DocumentDB Migration Tool:

  AccountEndpoint=https://lenadocumentdbdemo.documents.azure.com:443/;AccountKey=mLunZZcHSEZkHhRh6or902n3h9IMWu00jh6VdvrmaQBfb4HaWdOFjUZMLJqmloe5EbCc2aOWR70Oz2f91fWAQA==;**Database**=lenadocumentdbdemo;

- Click “Verify” icon.

Add data to your collection via Document DB migration tool

Upload json records to a Cosmos DB collection



- Create a collection by providing a name in the red box shown above (“migrationtoolupload” for e.g.). Partition key -empty; Collection Throughput- 400, add the name of Id Field (open json file and check, which is “id”). Accept defaults. Next, Next, Import.

Add data to your collection via Document DB migration tool

Upload json records to a Cosmos DB collection

Confirm import settings

Source (JSON file(s))

Files:	C:\Users\lena\Desktop\sampledata.json
Decompress data:	No

Target (DocumentDB - Sequential record import (partitioned collection))

Connection String:	AccountEndpoint=https:// lenadocumentdbdemo.documents.azure.com:443 =KrA71BQhL3Io1Z0IMlySqjCs7pGD6f3dPUg0IIKM s5zapoTOy8DiUm4cJF1MPyeCfXi72wORyTA==;D ocumentdbdemo;
Collection:	migrationtoolupload
Partition Key:	
Collection Throughput:	400
Id Field:	id
Number of Parallel Requests:	10

Import results

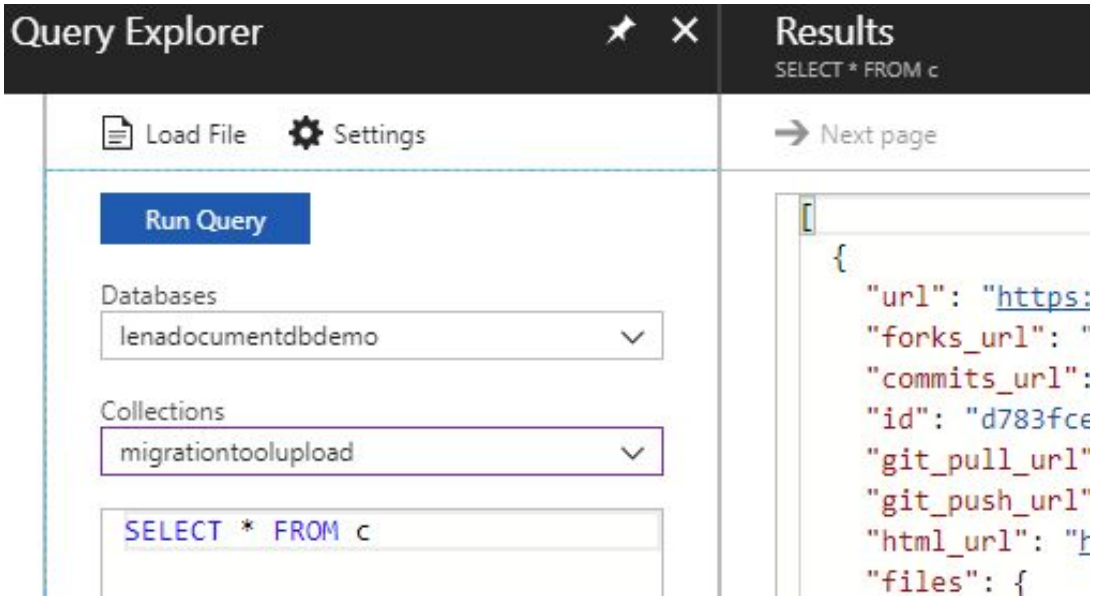
Elapsed time:	0:00:02.7
Transferred:	30
Failed:	0

All 30 records have been transferred.

Add data to your collection via Document DB migration tool

Upload json records to a Cosmos DB collection

Double-check in Azure portal by going into your collection, then either into **Data Explorer** or **Query Explorer**.

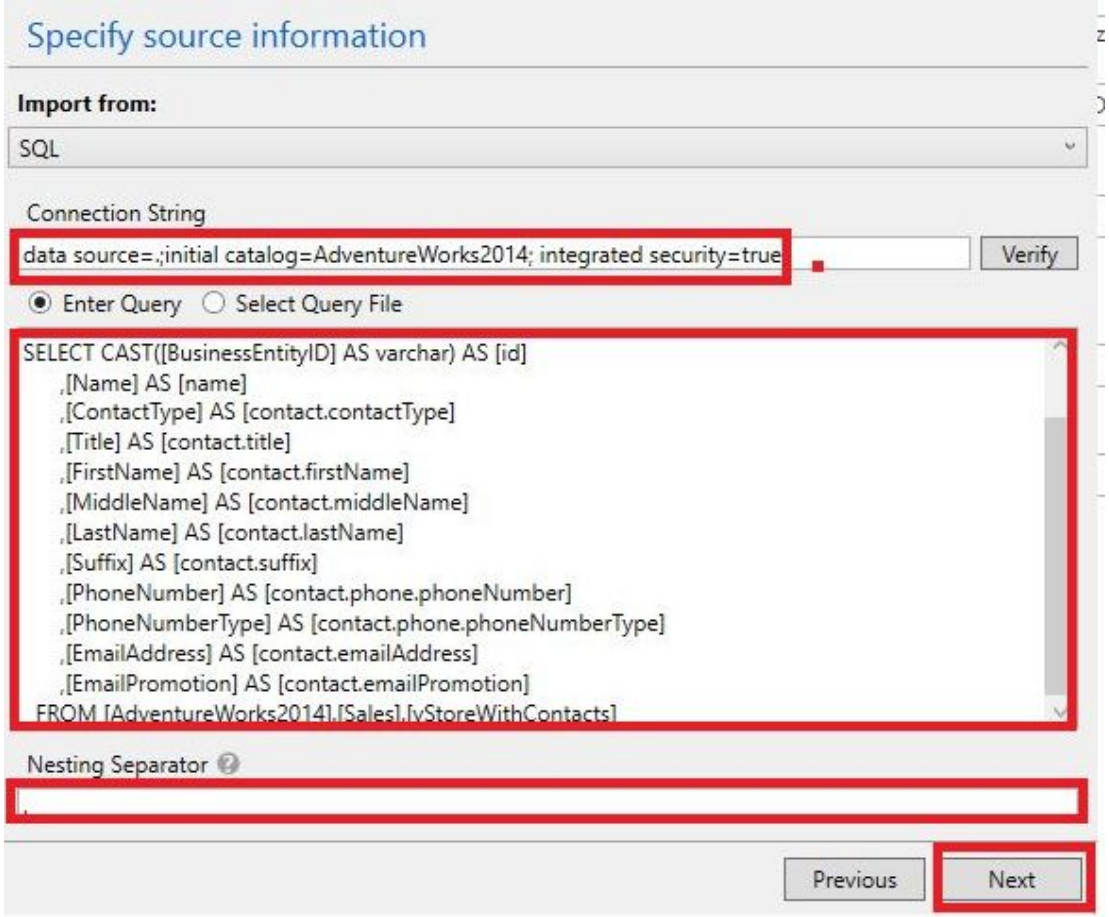


Add data to your collection via Document DB migration tool

Migrate SQL database to DocumentDB

Easy migration as shown in this tutorial:
https://www.devbition.com/migrate-sql-data-documentdb/

&
https://docs.microsoft.com/en-us/azure/cosmos-db/import-data



Build web app with .NET

In the lecture slides, we were shown creation of

- Mongo API Cosmos DB account
- Gremlin (graph) API Cosmos DB account

and execution of 2 Eclipse's Maven Java projects that used them as their backend, respectively.

For variety,

Let's create SQL API DB account and build .NET's C# project.

Build web app with .NET

Develop ASP.NET MVC application that connects to Azure's DocumentDB.

- Download the application's zip from https://github.com/Azure-Samples/documentdb-dotnet-todo-app or run in terminal: git clone https://github.com/Azure-Samples/documentdb-dotnet-todo-app.git

- Open in Visual Studio the **todo.sln** file coming from the unzipped 'documentdb-dotnet-todo-app' directory. (VS, File, Open, Project/solution, cd into documentdb-dotnet-todo-app, select todo.sln)

- Open **DocumentDBRepository.cs** file. On line 78, DocumentClient is initialized. It gets 'endpoint' and 'authKey' from web.config which we will get next.

Build web app with .NET

Code blocks of interest:

-Create DocumentClient:

```
78            client = new DocumentClient(new Uri(ConfigurationManager.AppSettings["endpoint"]),
79                                        ConfigurationManager.AppSettings["authKey"]);
```

-Create Database:

```
private static async Task CreateDatabaseIfNotExistsAsync()
{
    try
    {
        await client.ReadDatabaseAsync(UriFactory.CreateDatabaseUri(DatabaseId));
    }
    catch (DocumentClientException e)
    {
        if (e.StatusCode == System.Net.HttpStatusCode.NotFound)
        {
            await client.CreateDatabaseAsync(new Database { Id = DatabaseId });
        }
        else
        {
            throw;
        }
    }
}
```

Build web app with .NET

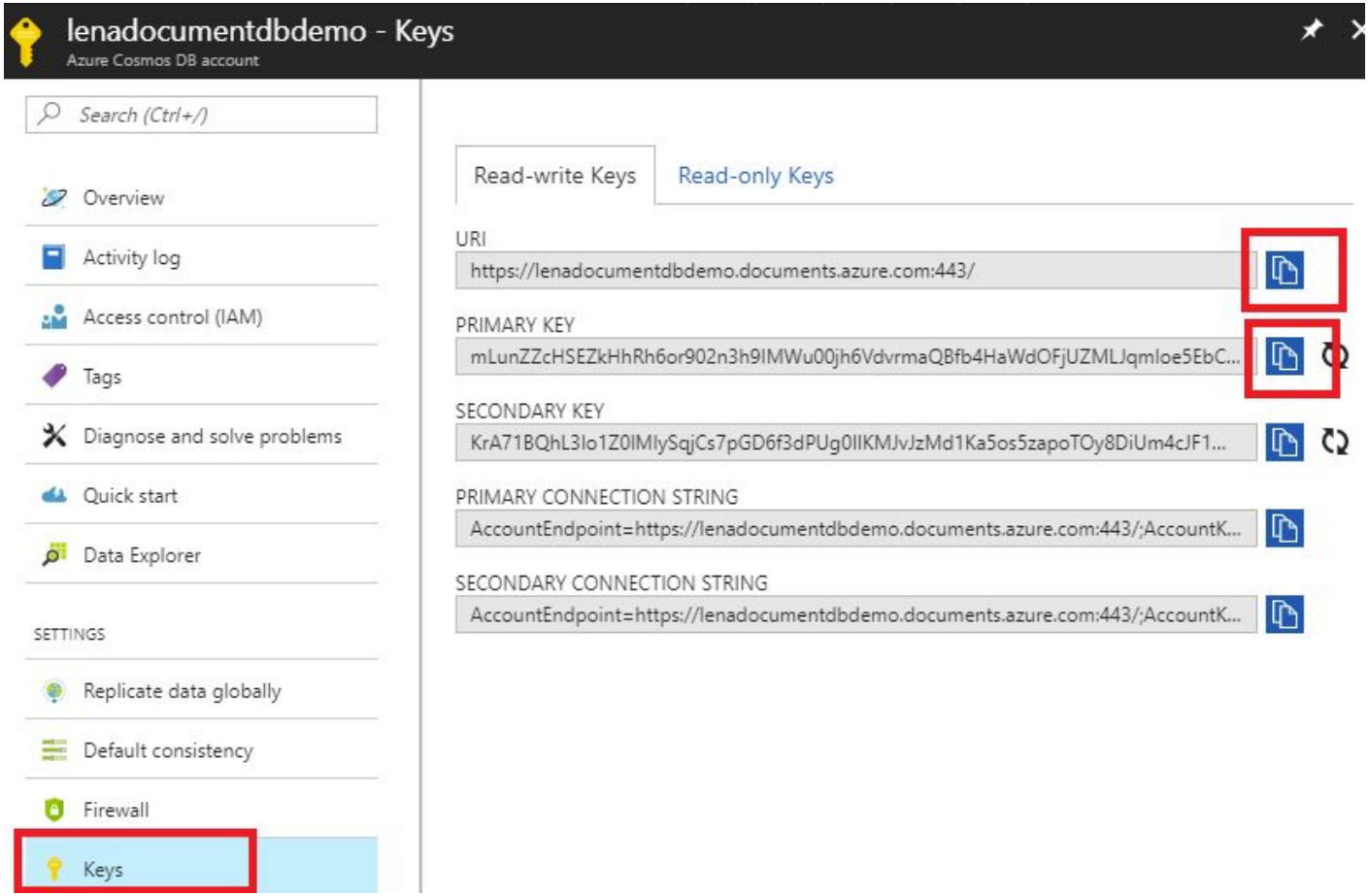
Code blocks of interest:

-Create Collection:

```
await client.CreateDocumentCollectionAsync(
    UriFactory.CreateDatabaseUri(DatabaseId),
    new DocumentCollection
        {
            Id = CollectionId,
            PartitionKey = new PartitionKeyDefinition() {
                                Paths = new Collection<string>() { "/category" } }
        },
    new RequestOptions { OfferThroughput = 1000 });
```

Build web app with .NET

- Get URI and the key from Azure Portal to update your connection string:



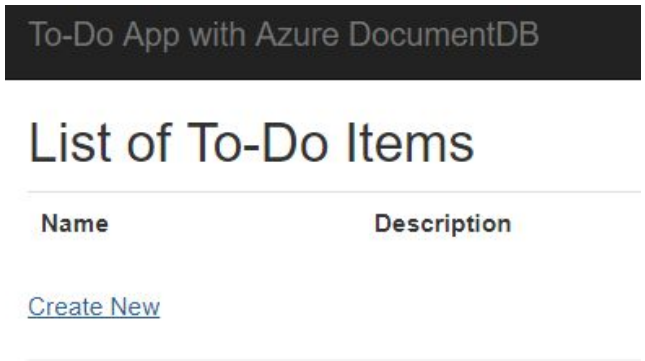
- Open **web.config** file in todo project and paste them there. Save.

Build web app with .NET

```
12      <add key="endpoint" value="https://lenadocumentdbdemo.document
13      <add key="authKey" value="mLunZZcHSEZkHhRh6or902n3h9IMWu00jh6V
```

- Right click on ‘todo’ project, **Manage NuGet Packages**, type ‘**DocumentDB**’, install.

- Build, Build Solution. Select ‘todo’ project, press **Ctrl + F5** to run it. The app is displayed in the browser at http://localhost:43605/ . Click on ‘Create New’ and create a few records.



- Now you can query, modify, and work with this new data in Azure portal via **Data Explorer** or **Query Explorer.**

Build web app with .NET

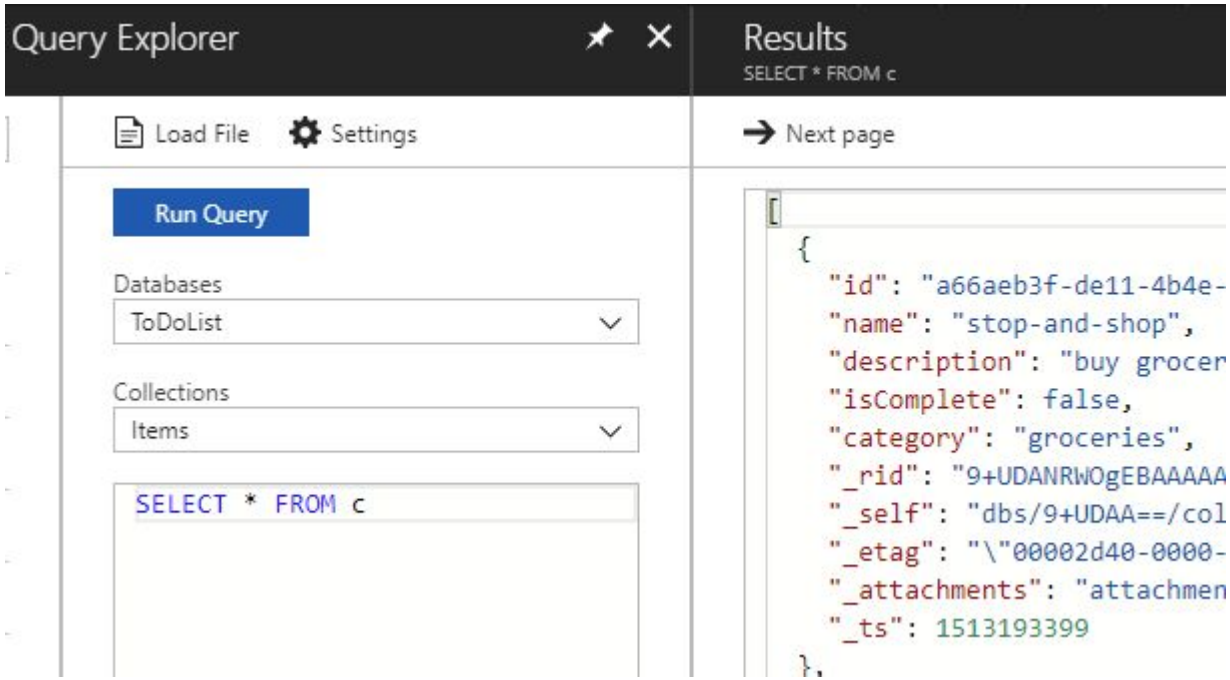
--Added a few records:

List of To-Do Items

Name	Description	Category
stop-and-shop	buy groceries today	groceries
walmart	buy lights and X-mas tree ornaments	xmas stuff
whole foods	buy 1 bar of the best soup ever and get out	beauty
cvs	exchange batteries since bought wrong ones	fix watch

Create New

--Queried in Portal:



Build web app with .NET

--Or query in the application itself:

https://docs.microsoft.com/en-us/azure/cosmos-db/documentdb-sql-query

for e.g:

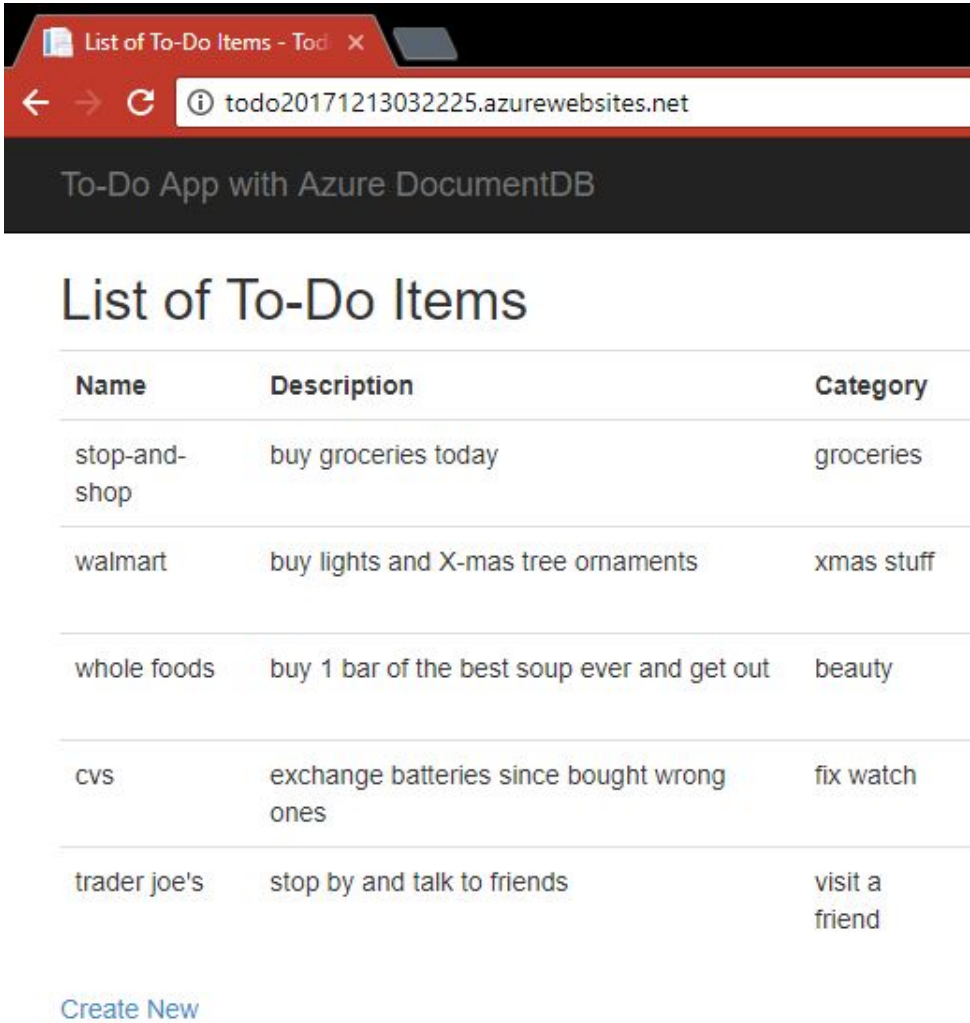
```
foreach (var family in client.CreateDocumentQuery(collectionLink,
    "SELECT * FROM Families f WHERE f.id = \"AndersenFamily\""))
{
    Console.WriteLine("\tRead {0} from SQL", family);
}

SqlQuerySpec query = new SqlQuerySpec("SELECT * FROM Families f WHERE f.id = @familyId");
query.Parameters = new SqlParameterCollection();
query.Parameters.Add(new SqlParameter("@familyId", "AndersenFamily"));

foreach (var family in client.CreateDocumentQuery(collectionLink, query))
{
    Console.WriteLine("\tRead {0} from parameterized SQL", family);
}
```

--Also take a look at the last section there. Cosmos DB provides a programming model for executing JavaScript-based application logic directly on the collections using stored procedures and triggers.

Build web app with .NET

- Publish your app to Azure:

-Right Click on “todo” project, Publish,
-Select ‘Microsoft Azure App Services,’ Create New,
-Re-authenticate with Microsoft,
-Fill out app’s information
(I’ve kept the app name randomly generated by Azure for me: todo20171213032225),
-Create, then wait until the browser tab with the deployed application opens.



Build web app with .NET

A little recap from previous weeks:

You can also publish in different ways. To deploy from local git, for example, read the below instructions: https://docs.microsoft.com/en-us/azure/app-service/app-service-deploy-local-git

In the same section, you can find instructions to deploy from ARM Templates, MsBuild, PowerShell, Web Deploy, and etc.

Create DB, collection and populate it with data via Python SDK

Demo time: see **cosmosDBdemo.ipnb**

***I am using python 3.4 and the machine + setup as described in Lab 7, slide 9.

***Do not forget to delete the resource group for this lab (‘documdbrg’ in my case with all of the resources in it as shown below).

☐	lenadocumentdbdemo	Azure Cosmos DB account
☐	todo20171213032225	App Service
☐	todo20171213032225Plan	App Service plan

Create DB, collection via bash script

Run **script1.sh** to create a **SQL API** Cosmos DB account, DB in it and a collection within the later, to scale down collection's throughput, and to list the SQL API gloabalDBAccount URI and primary key. (The last two can be used instead of manually looking through Azure Portal as we did with .NET application we've developed in the previous slides.)

az cosmosdb update

Update an Azure Cosmos DB database account.

Azure CLI

```
az cosmosdb update --name
                   --resource-group
                   [--default-consistency-level {BoundedStaleness,
                   [--enable-automatic-failover]
                   [--ip-range-filter]
                   [--locations]
                   [--max-interval]
                   [--max-staleness-prefix]
```

Create DB, collection via bash script

Options to consider while create a database:

`--kind`

The type of Cosmos DB database account to create.
accepted values: GlobalDocumentDB, MongoDB, Parse
default value: GlobalDocumentDB

`--locations`

Space separated locations in 'regionName=failoverPriority' format. E.g "East US"=0 "West US"=1. Failover priority values are 0 for write regions and greater than 0 for read regions. A failover priority value must be unique and less than the total number of regions. Default: single region account in the location of the specified resource group.

`--default-consistency-level`

Default consistency level of the Cosmos DB databas
accepted values: BoundedStaleness, ConsistentPrefix, Event

`--enable-automatic-failover`

Enables automatic failover of the write region in the rare event that the region is unavailable due to an outage. Automatic failover will result in a new write region for the account and is chosen based on the failover priorities configured for the account.

`--ip-range-filter`

Firewall support. Specifies the set of IP addresses or IP address ranges in CIDR form to be included as the allowed list of client IPs for a given database account. IP addresses/ranges must be comma separated and must not contain any spaces.

Create DB, collection via bash script

Scaling a collection's throughput; code from **script1.sh**.

```
#Scale throughput
az cosmosdb collection update \
    --collection-name $collectionName \
    --name $name \
    --db-name $databaseName \
    --resource-group $resourceGroupName \
    --throughput $newThroughput
```

Define 'throughput': Loosely, it means the number of transactions per second. Keep in mind that "write" transactions are different than "read" transactions, and sustained rates are different than peak rates.

We need to reserve throughput in Azure Cosmos DB so it is available to your application on per second basis by specifying how many **request units** to process **per second**. Each operation in Azure Cosmos DB - writing a document, performing a query, updating a document - consumes CPU, memory, and IOPS. That is, each operation incurs a request charge, which is expressed in request units. https://docs.microsoft.com/en-us/azure/cosmos-db/request-units

Create DB, collection via bash script

Define 'partitioning': instrument that allows a collection massively scale in terms of storage and throughput needs.

A **partition hosts one or more partition keys**, a **collection** acts as the **logical container of these physical partitions**. Documents with the same partition key are grouped together always in the same physical partition. If you pick **DateCreated** partition key, all pictures uploaded on the same date would be in the same partition. Picking **Category** partition, all flowers will be in a Flower partition.

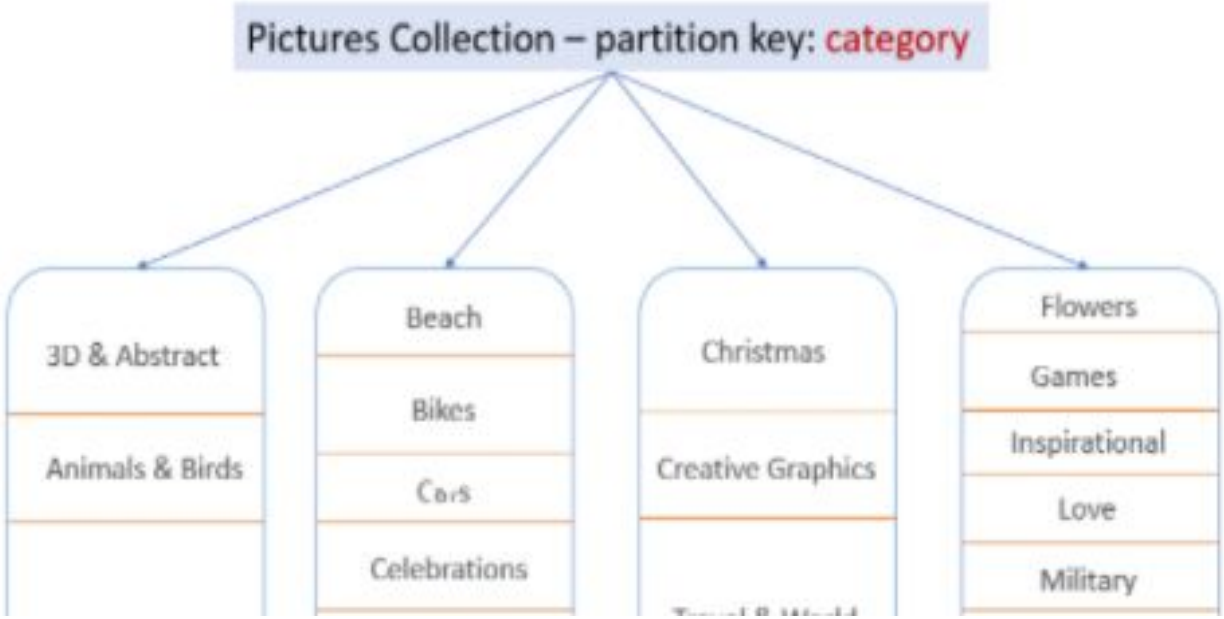


Image of 4 collections with various partitions-categories.

Create DB, collection via bash script

Console's output after running **script1.sh**:

```
buntu@ubuntu:$ sh script1.sh
Creating RG...
    "id": "/subscriptions/c889f1c0-5c27-4209-a79c
    "name": "documdbrg",
Success.
-----------------------------------------------------

Creating SQL API GlobalDB-DocumentDB account...
  "provisioningState": "Succeeded",
      "provisioningState": "Succeeded"
      "provisioningState": "Succeeded"
      "provisioningState": "Succeeded"
Success.
-----------------------------------------------------

Creating a database in it...
"test-docdb"
Success.
-----------------------------------------------------

Creating a collection within this database...
    "id": "docdbcollection1",
Success.
-----------------------------------------------------

Updating this collection's throughput to a new va
Success.
-----------------------------------------------------

Show this globalDBAccount URI ...
    "https://lenadocumentdbdemo.documents.azure.c

Show this globalDBAccount primary key...
  "primaryMasterKey": "tobtttcpgq7wuBotIKWYlRjJ7K
",
-----------------------------------------------------
Done.
```

Create DB, collection via bash script

Run **script2.sh** to create
a **Mongo API** Cosmos DB account,
DB in it and a collection (s) in it,
to update collection's throughput,
and to list mongoAccount connection string
so it can be used for MongoDB apps.

Console's output:

```
Creating a MongoDB API Cosmos DB account
  "provisioningState": "Succeeded",
      "provisioningState": "Succeeded"
      "provisioningState": "Succeeded"
Success.
----------------------------------------

Creating a database in it...
"test-mongodb"
Success.
----------------------------------------

Creating a collection within the databas
    "id": "mongodbcollection1",
Success.
----------------------------------------

Updating this collection's throughput to
Success.
----------------------------------------

This MongoAPICosmosDBAccount connection
{
  "connectionStrings": [
    {
      "connectionString": "mongodb://len
r1F2f5GyCijHH5MpNYTjdZo3O2ZWSjWH82b315qM
cuments.azure.com:10255/?ssl=true",
      "description": "Default MongoDB Co
    }
  ]
}
----------------------------------------
Done.
```

Create DB, collection via bash script

Cosmos DB CLI reference to help with creating bash scripts:
https://docs.microsoft.com/en-us/cli/azure/cosmosdb?view=azure-cli-latest

CosmosDB and Azure Functions for Serverless architecture

With the native integration between Azure Cosmos DB and Azure Functions, you can:

- in particular, create database triggers, input bindings, and output bindings directly from your Azure Cosmos DB account.
- in general, create and deploy **event-driven serverless apps** with low-latency access to rich data for a global user base.

Integrate your databases and serverless apps via

- **bindings** (bind an Azure Function to an Azure Cosmos DB collection using an input or output bindings which either read data or write data to a container when a function executes.
  or
- **triggers** (event-driven CosmoDB triggers which rely on **Change Feed** streams to monitor your Azure Cosmos DB container, and when a change/modification to it happens, the Change Feed stream is sent to the trigger, which invokes the Azure Function.)

CosmosDB and Azure Functions for Serverless architecture

Important concept: Change Feed

To help you build powerful applications on top of Cosmos DB, Azure have built Cosmos DB **Change Feed**, which is essentially **a persistent log of records in the order which they were modified.** It includes inserts and updates made to a document (deletes also can be captured) and is sorted in the order of modification within each partition key value.

Using Change Feed and Azure Functions you can implement a microservice for your application; perform stream processing using Spark, for example; and archive data, storing hot data (=frequently-accessed data) in Cosmos DB and cold data in Azure Data Lake or Storage Table.

CosmosDB and Azure Functions for Serverless architecture

Important concept: Azure Functions

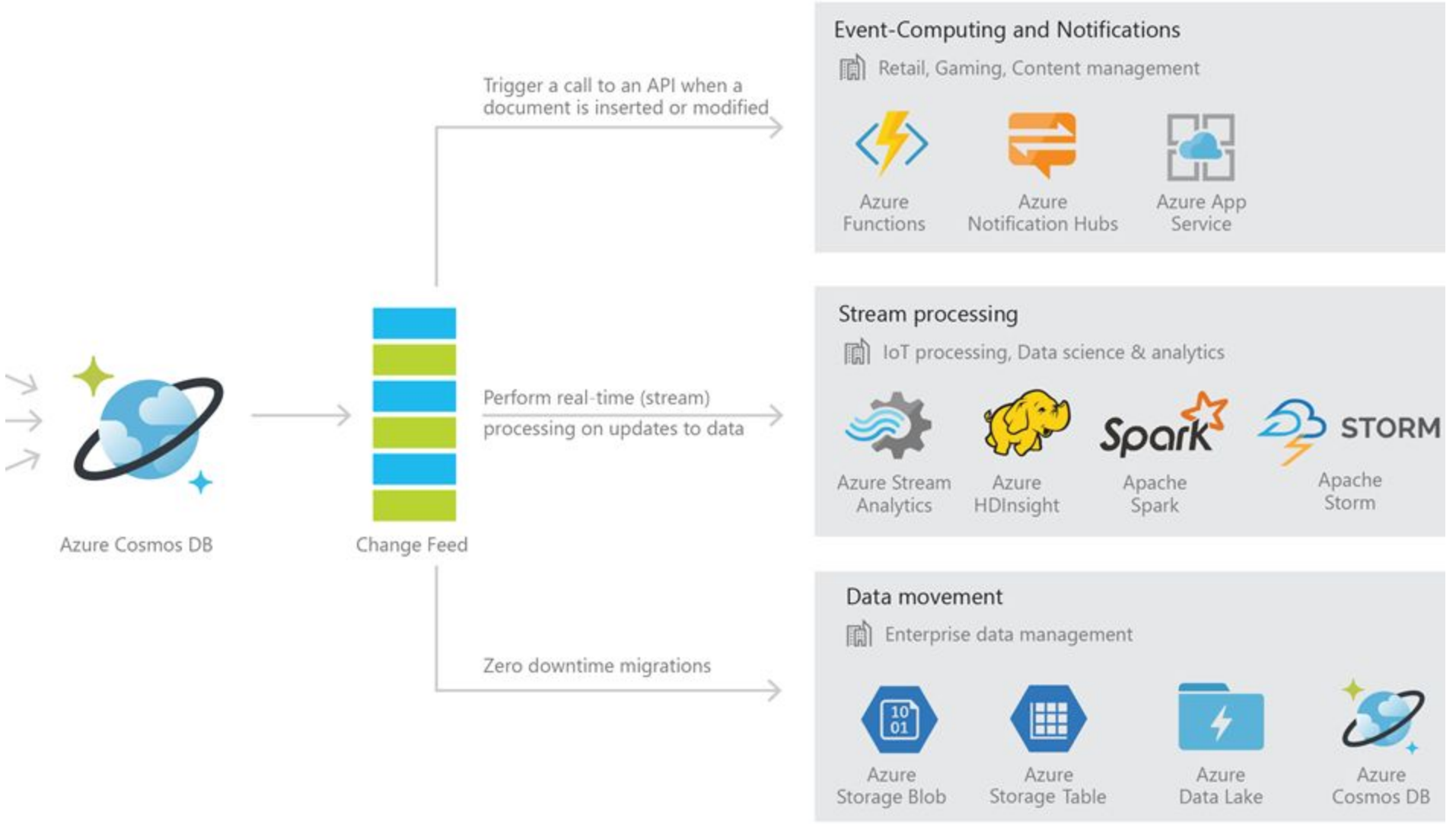
Using Azure Functions,

- we don't have to create a full-blown app to respond to changes in our Azure Cosmos DB database, instead creating small reusable functions for specific tasks.

- we can use Azure Cosmos DB data as the input or output to an Azure Function in response to event such as an HTTP requests or a timed trigger.

- we can perform tasks quickly. The service spins up new instances of functions whenever an event fires and closes them as soon as the function completes. Users only pay for the time their functions are running.

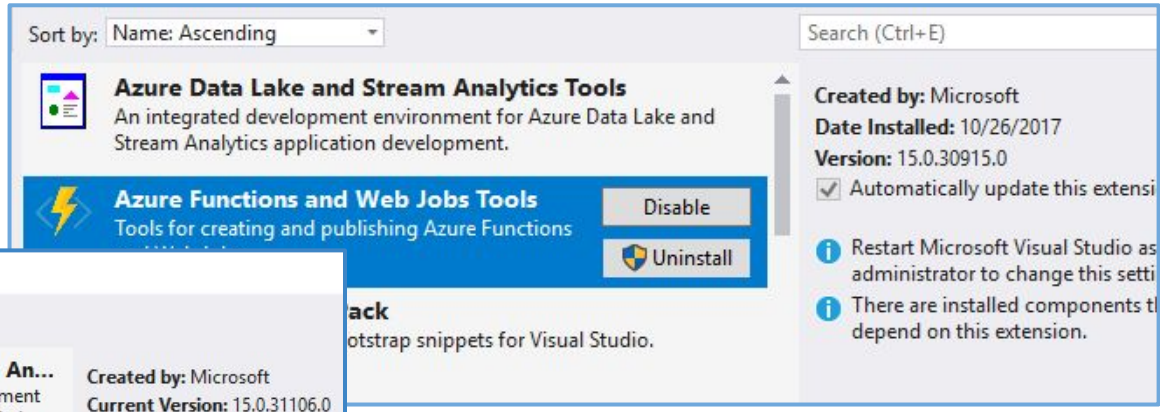
CosmosDB and Azure Functions for Serverless architecture

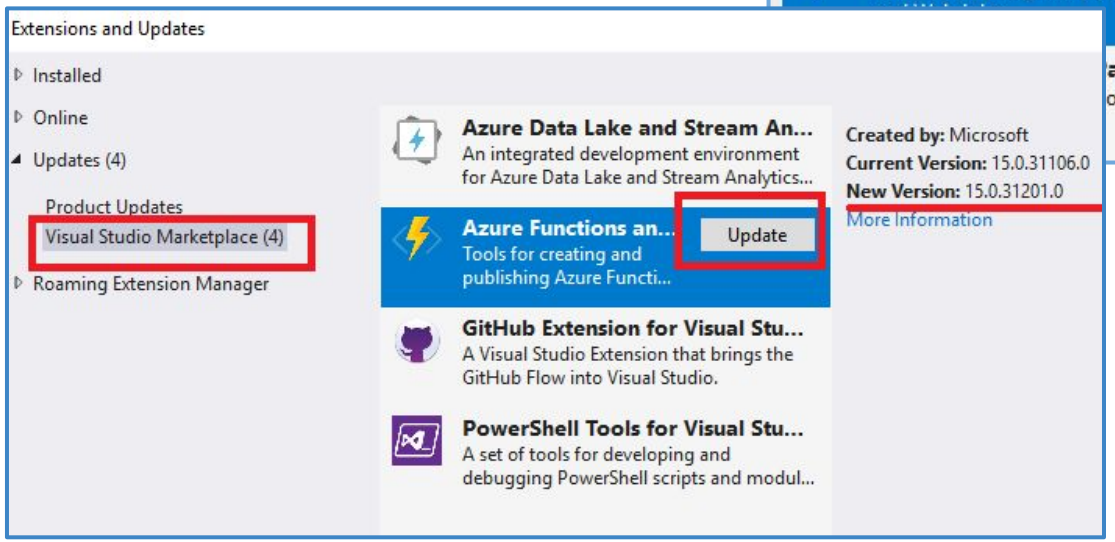


CosmosDBTrigger Project

<u>Create CosmosDBTrigger, which is a function, to monitor changes to a Cosmos DB container</u>

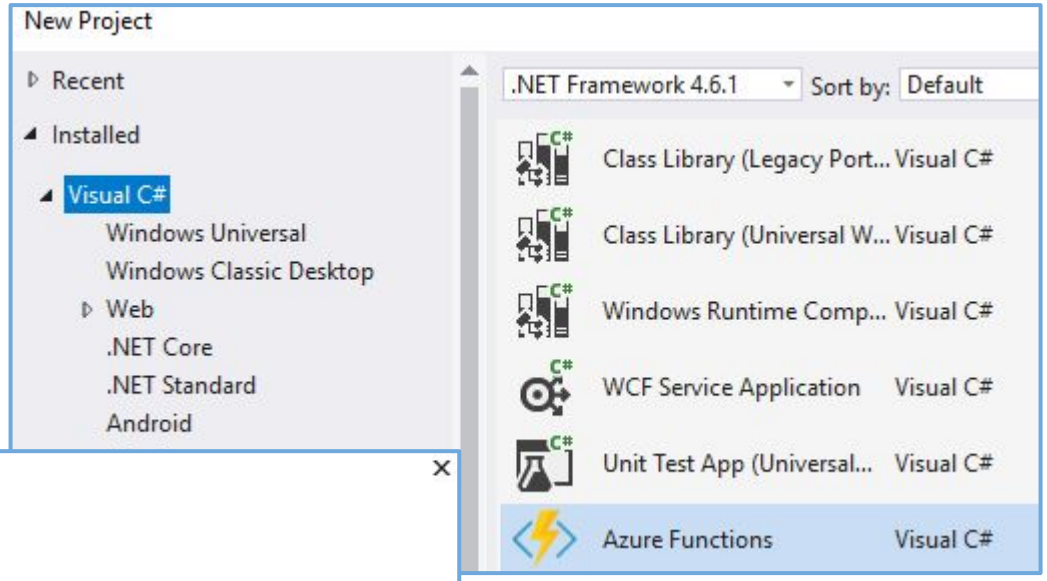
- Open VS 2017
- Tools, Extensions and Updates, check if “Azure Functions and Web Jobs Tool’s” version = 15.0.31201.0. If your version is lower, update it, running VS Studio 2017 as an admin.

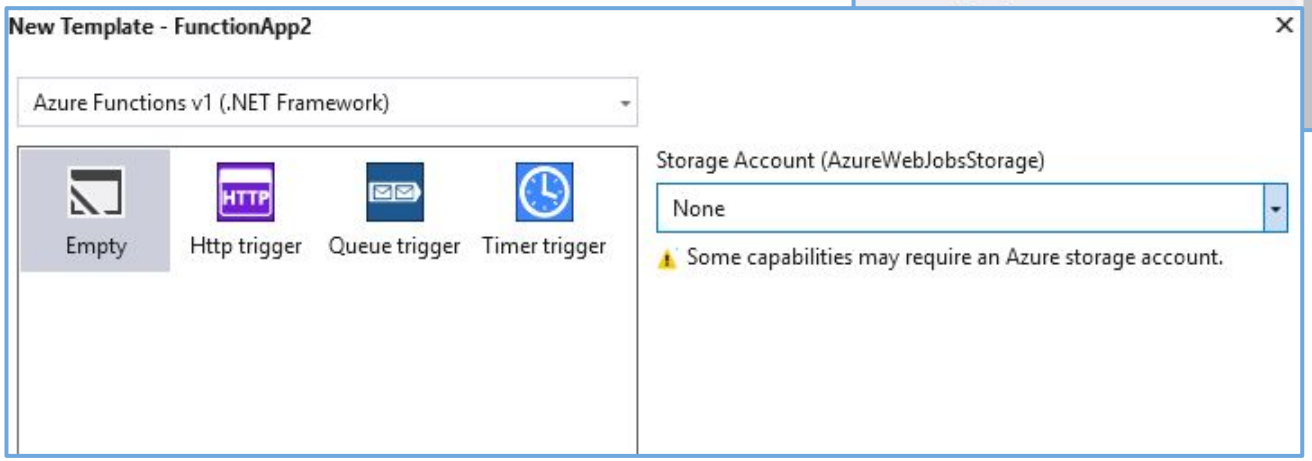




CosmosDBTrigger Project

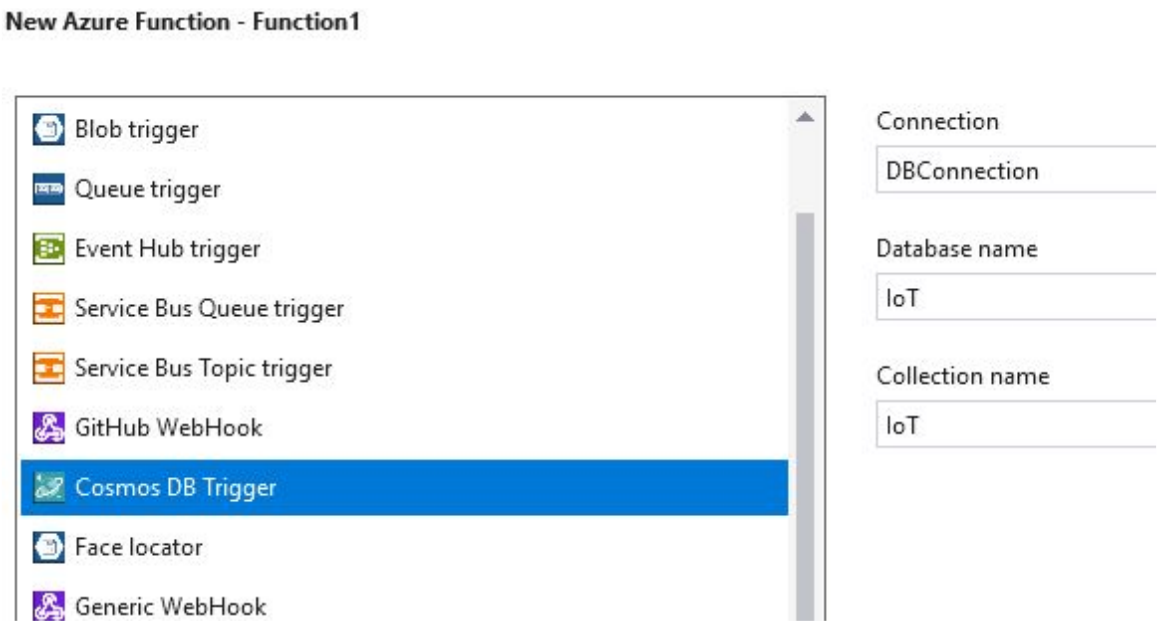
File, New, Project, Visual C#, Azure Functions. Name this project: ‘FunctionApp1’, keep other defaults, Ok. Select Empty, in the dropdown meny have v1(.NET Framework), in the Storage Account dropdown, select None. OK.





CosmosDBTrigger Project

- Click on ‘FunctionApp1’ project, Add, New Item, Azure Function, accept default name, choose ‘Cosmos DB Trigger’. Connection: name it ‘DBConnection’, Database Name: IoT, Collection name: IoT. Ok.



- Open Function1.cs and examine it:

CosmosDBTrigger Project

```
public static void Run([CosmosDBTrigger(
    databaseName: "IoT",
    collectionName: "IoT",
    ConnectionStringSetting = "DBConnection",
    LeaseCollectionName = "leases")]IReadOnlyList<Document> input, TraceWriter log)
{
    if (input != null && input.Count > 0)
    {
        log.Verbose("Documents modified " + input.Count);
        log.Verbose("First document Id " + input[0].Id);
    }
}
```

→ Copy ConnectionStringSettings's value which is "DBConnection" into settings file: **local.settings.json**, in particular, add this line to the body of the json: "DBConnection": ""

CosmosDBTrigger Project

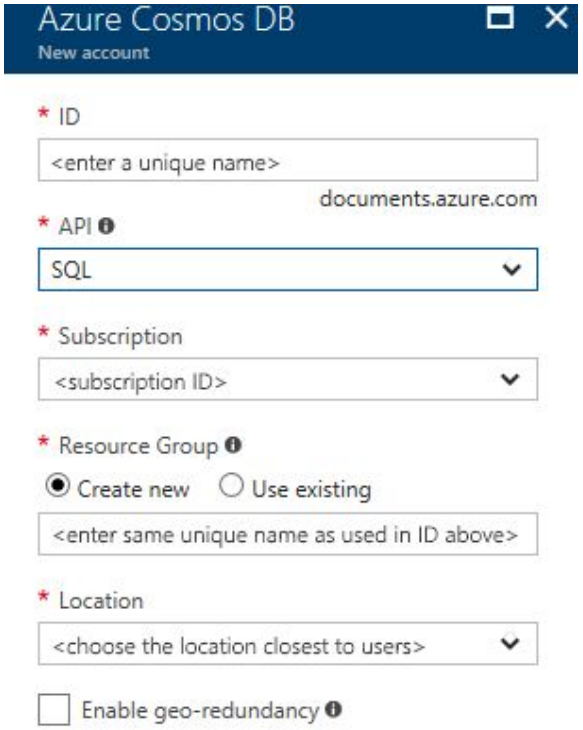
- Next, fill out the 3 missing values in **local.settings.json**

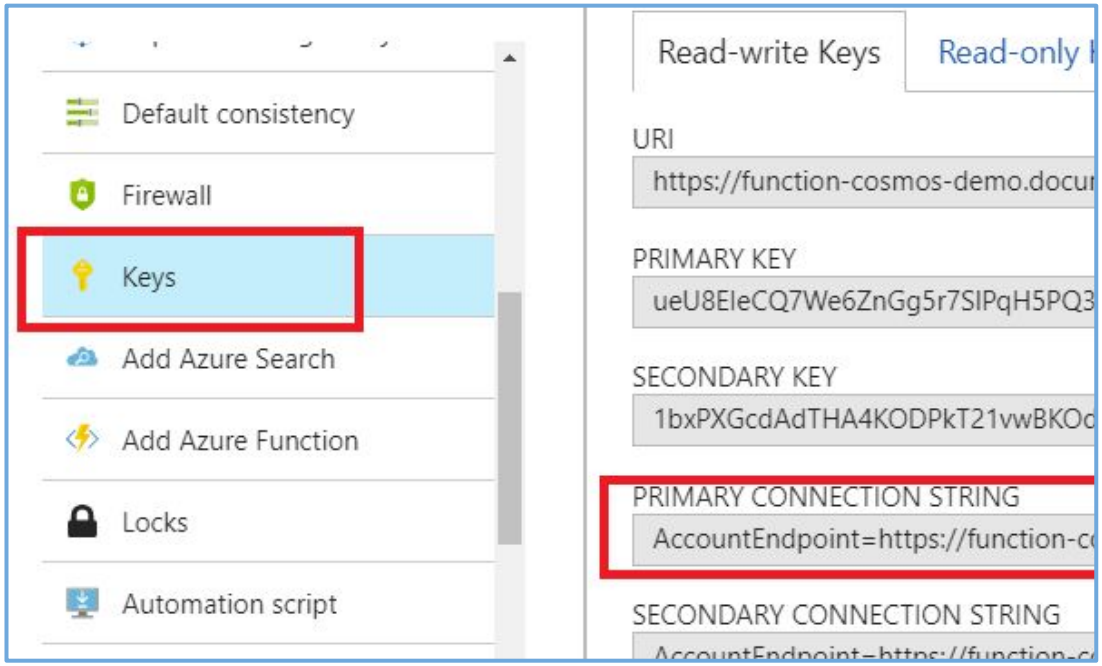
```
{
    "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "",
    "AzureWebJobsDashboard": "",
    "DBConnection":  ""
  }
}
```

1. **DBConnection** which is obtained after creating Cosmos DB account from which we will copy the connection string;
2. **AzureWEbJobsStorage** which is obtained from a storage account, Access keys, copy 1st connection string;
3. **AzureWEbJobsDashboard** which is obtained from a storage account, Access keys, copy 2nd connection string.

CosmosDBTrigger Project

1. Click In the Portal, create SQL API cosmos DB account named “function-cosmos-demo”, res group: “functionrg." Then go to Keys, and copy Primary Connection String pasting it in local.settings.json

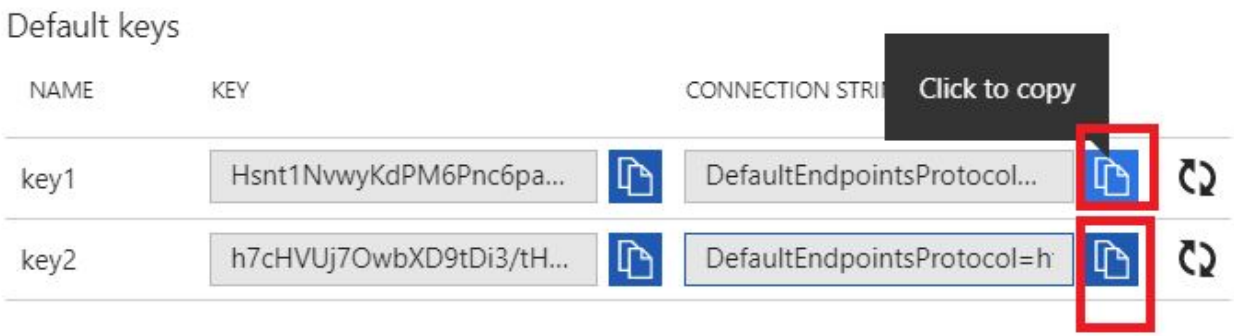




***Also create 2 needed-later collections (“IoT” and “leases”) that belongs to “IoT” database.

CosmosDBTrigger Project

2,3. Now create a storage account: Click on ‘Storage Accounts’ in the Portal’s left panel, +Add. Name: lenastorageforfunction, Cool, Existing resource group: functionrg. Then go to Access Keys and copy the Connection Strings pasting them into local.settings.json:



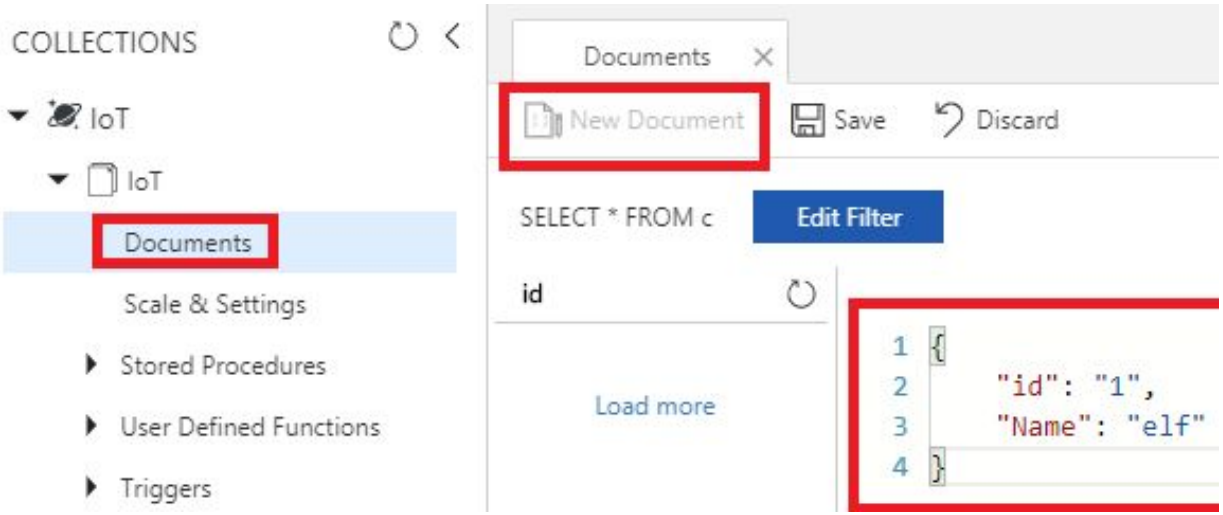
→ Back in VS, run ‘FunctionApp1’ project. If needed, install Function CLI tools and enable them in Firewall. Check in cmd applet that everything works.

CosmosDBTrigger Project

Function App is running indeed:

```
C:\WINDOWS\system32\cmd.exe
[12/14/2017 3:34:56 PM] }
[12/14/2017 3:34:56 PM] Executed HTTP request: {
[12/14/2017 3:34:56 PM]   "requestId": "43178063-5f
[12/14/2017 3:34:56 PM]   "method": "GET",
[12/14/2017 3:34:56 PM]   "uri": "/",
[12/14/2017 3:34:56 PM]   "authorizationLevel": "Ar
[12/14/2017 3:34:56 PM] }
[12/14/2017 3:34:56 PM] Response details: {
[12/14/2017 3:34:56 PM]   "requestId": "43178063-5f
[12/14/2017 3:34:56 PM]   "status": "OK"
[12/14/2017 3:34:56 PM] }
Debugger listening on [::]:5858
[12/14/2017 3:34:58 PM] Job host started
```

Set a breakpoint on line 19 in **Function1.cs**. Then Insert a sample record in "IoT" Db, "IoT" collection. Save.



CosmosDBTrigger Project

Cmd applet (=Change Feed) confirms there was a change to collection named “IoT” and triggers our Function to be executed.

In VS, hover over **input.Count** variable to see its value:

```
18  {
19      log.Verbose("Documents modified " + input.Count);
20      log.Verbose("First document Id " + input
21  }
22
```

input Count = 1

```
18  {
19      log.Verbose("Documents modified " + input.Count);
20      log.Verbose("First document Id " + input[
21  }
22
23  }
```

[0] {{ "id": "1", "Name": "elf", "_rid": "3yJdAK8nEwAHAAAAAAAAAA==", "_self": "dbs/3yJdA

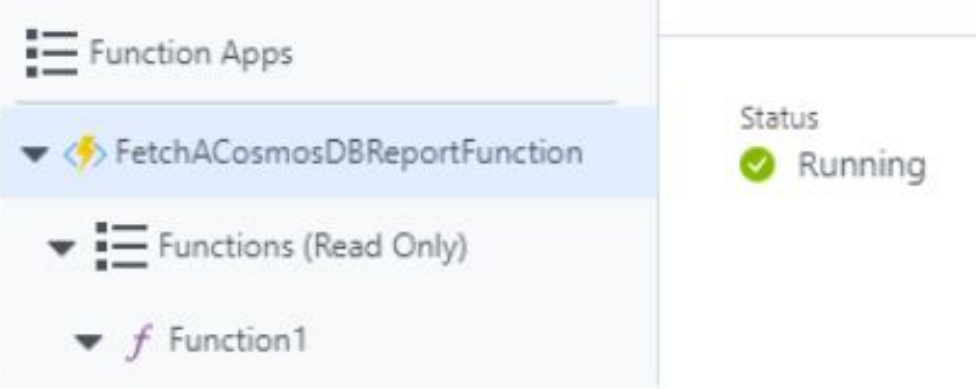
Raw View

The last few slides have demonstrated how Cosmos DB, Azure Functions and VS 2017 can all work nicely together.

CosmosDBTrigger Project

- Publish this function in Azure Portal:
  Right click on ‘FunctionApp1’ project, Publish, (accept defaults- Create New Azure Function App). Publish. App Name: ‘CosmosDBTriggerFunction’... Create.

- Check in Portal that our function app has been successfully published:



Miscellaneous

We just saw an example of a Cosmos DB trigger. If you would like to also take a look at Cosmos DB bindings, you can follow this link: https://www.jan-v.nl/post/use-bindings-with-azure-functions

If you have time, as an exercise, please try migrating an SQL database via DocumentDB Migration Tool. How easy/difficult/time-consuming is this operation? Does it make sense? If you were to migrate an SQL database to Cosmos DB which additional, maybe better, tools you might have used? Some questions to research...

Thank you!