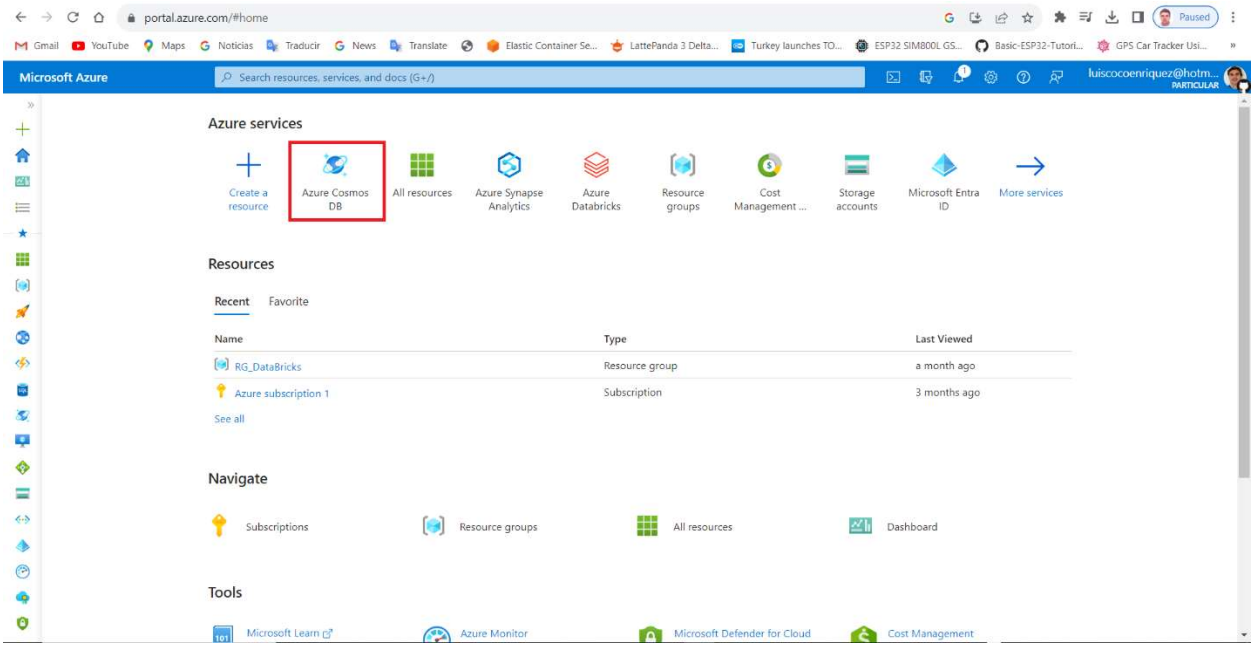
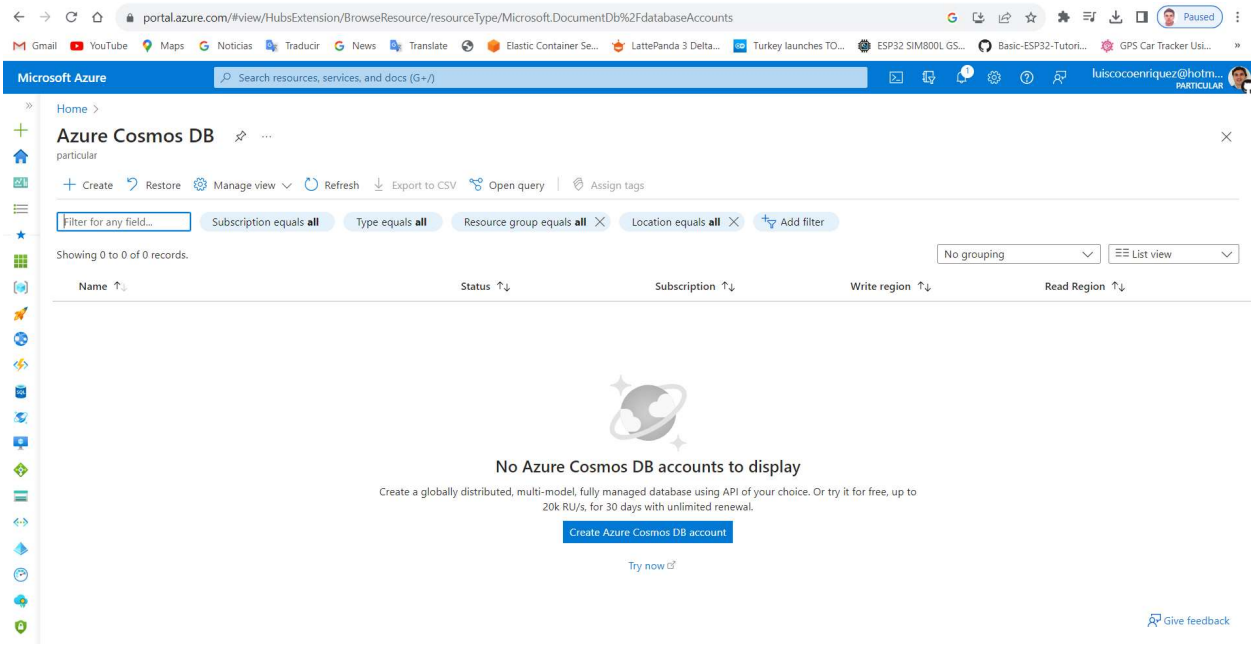
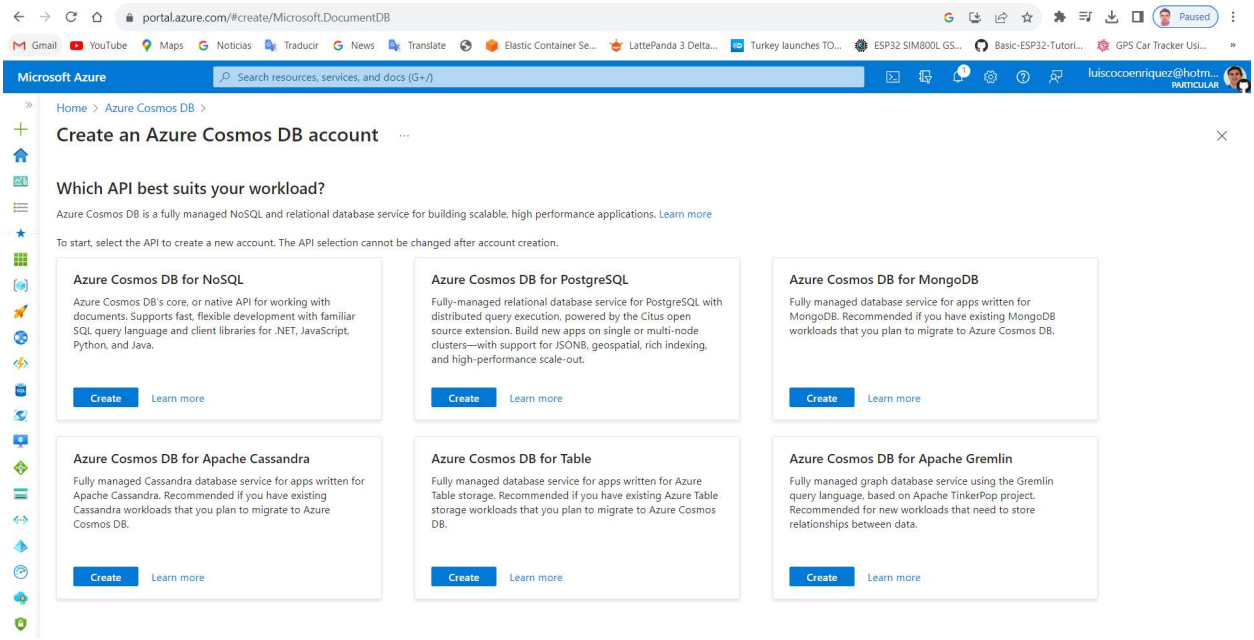
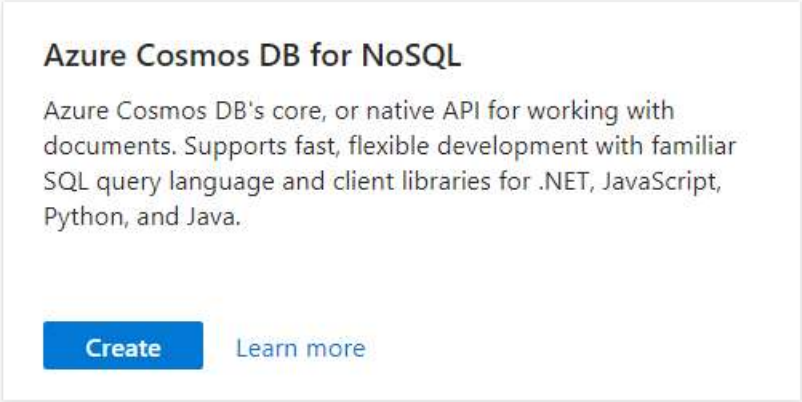
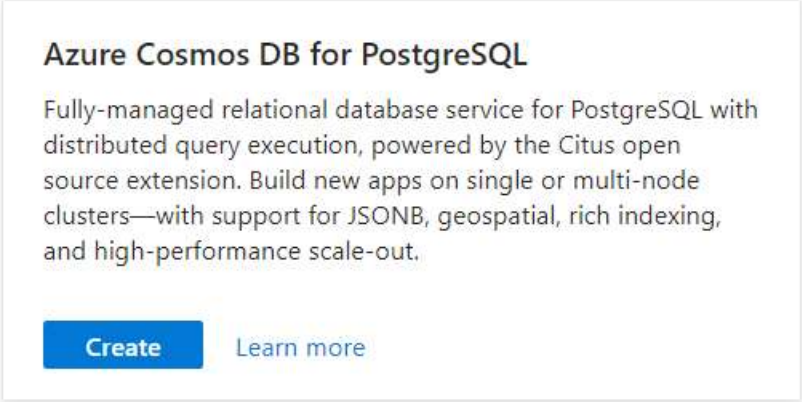
Azure CosmosDB (samples with .NET, Java, Node.js and Python)






portal.azure.com/#create/Microsoft.DocumentDB
Microsoft Azure
Search resources, services, and docs (G+/)
luiscocoenriquez@hotm...
PARTICULAR
Home > Azure Cosmos DB >
Create an Azure Cosmos DB account
Which API best suits your workload?
Azure Cosmos DB is a fully managed NoSQL and relational database service for building scalable, high performance applications. Learn more
To start, select the API to create a new account. The API selection cannot be changed after account creation.
Azure Cosmos DB for NoSQL
Azure Cosmos DB's core, or native API for working with documents. Supports fast, flexible development with familiar SQL query language and client libraries for .NET, JavaScript, Python, and Java.
Create
Learn more
Azure Cosmos DB for PostgreSQL
Fully-managed relational database service for PostgreSQL with distributed query execution, powered by the Citus open source extension. Build new apps on single or multi-node clusters—with support for JSONB, geospatial, rich indexing, and high-performance scale-out.
Create
Learn more
Azure Cosmos DB for MongoDB
Fully managed database service for apps written for MongoDB. Recommended if you have existing MongoDB workloads that you plan to migrate to Azure Cosmos DB.
Create
Learn more
Azure Cosmos DB for Apache Cassandra
Fully managed Cassandra database service for apps written for Apache Cassandra. Recommended if you have existing Cassandra workloads that you plan to migrate to Azure Cosmos DB.
Create
Learn more
Azure Cosmos DB for Table
Fully managed database service for apps written for Azure Table storage. Recommended if you have existing Azure Table storage workloads that you plan to migrate to Azure Cosmos DB.
Create
Learn more
Azure Cosmos DB for Apache Gremlin
Fully managed graph database service using the Gremlin query language, based on Apache TinkerPop project. Recommended for new workloads that need to store relationships between data.
Create
Learn more


Azure Cosmos DB for NoSQL
Azure Cosmos DB's core, or native API for working with documents. Supports fast, flexible development with familiar SQL query language and client libraries for .NET, JavaScript, Python, and Java.
Create
Learn more


Azure Cosmos DB for PostgreSQL
Fully-managed relational database service for PostgreSQL with distributed query execution, powered by the Citus open source extension. Build new apps on single or multi-node clusters—with support for JSONB, geospatial, rich indexing, and high-performance scale-out.
Create
Learn more

Azure Cosmos DB for MongoDB

Fully managed database service for apps written for MongoDB. Recommended if you have existing MongoDB workloads that you plan to migrate to Azure Cosmos DB.

Create Learn more

Azure Cosmos DB for Apache Cassandra

Fully managed Cassandra database service for apps written for Apache Cassandra. Recommended if you have existing Cassandra workloads that you plan to migrate to Azure Cosmos DB.

Create Learn more

Azure Cosmos DB for Table

Fully managed database service for apps written for Azure Table storage. Recommended if you have existing Azure Table storage workloads that you plan to migrate to Azure Cosmos DB.

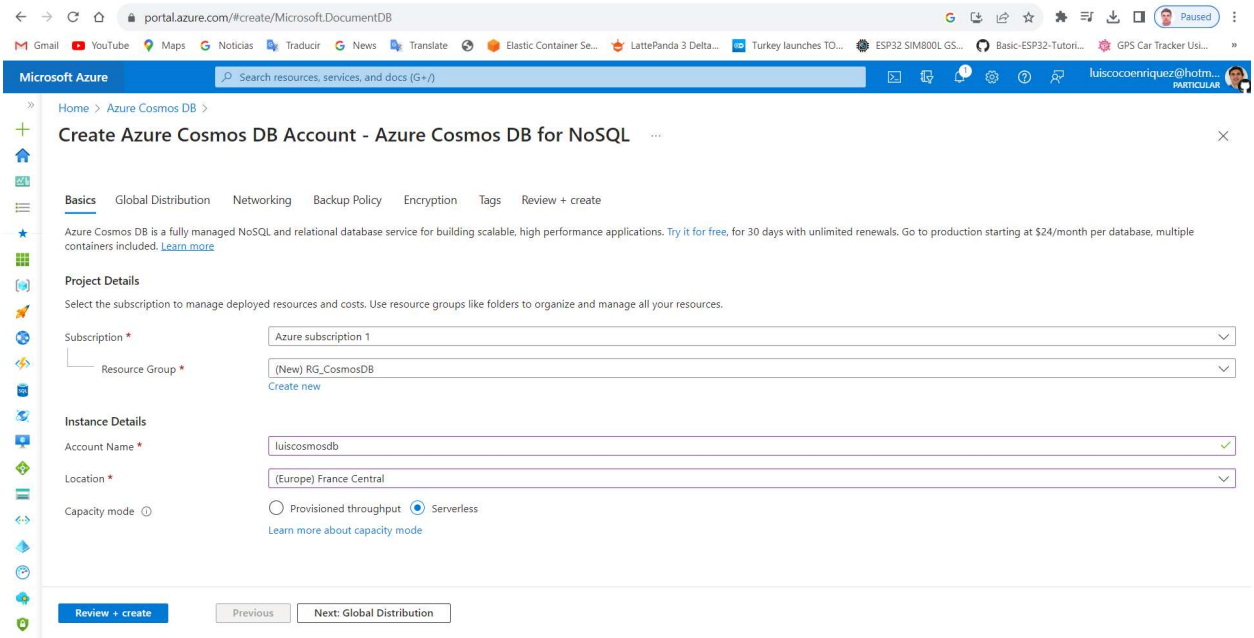
Create Learn more

Azure Cosmos DB for Apache Gremlin

Fully managed graph database service using the Gremlin query language, based on Apache TinkerPop project. Recommended for new workloads that need to store relationships between data.

Create Learn more

Azure Cosmos DB for NoSQL



portal.azure.com/#create/Microsoft....

Gmail YouTube Maps Noticias Traducir News Translate Elastic Container Se...

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB fo...

Basics | Global Distribution | Networking | Backup Policy | Encryption | Tags

Azure Cosmos DB is a fully managed NoSQL and relational database service for building scalable, high performance applications. Try it for free, for 30 days with unlimited renewals. Go to production starting at $24/month per database, multiple containers included. Learn more

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Azure subscription 1

Resource Group * (New) RG_CosmosDB

Create new

Instance Details

Account Name * luiscosmosdb

Location * (Europe) France Central

Capacity mode ⓘ ○ Provisioned throughput ◉ Serverless

Learn more about capacity mode

Review + create | Previous | Next: Global Distribution

portal.azure.com/#create/Microsoft.Docu...

Gmail YouTube Maps Noticias Traducir News Translate Elastic Container Se...

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB fo...

Subscription * Azure subscription 1

Resource Group * (New) RG_CosmosDB

Create new

Instance Details

Account Name * luiscosmosdb

Location * (Europe) France Central

Capacity mode ⓘ Provisioned throughput Serverless

Learn more about capacity mode

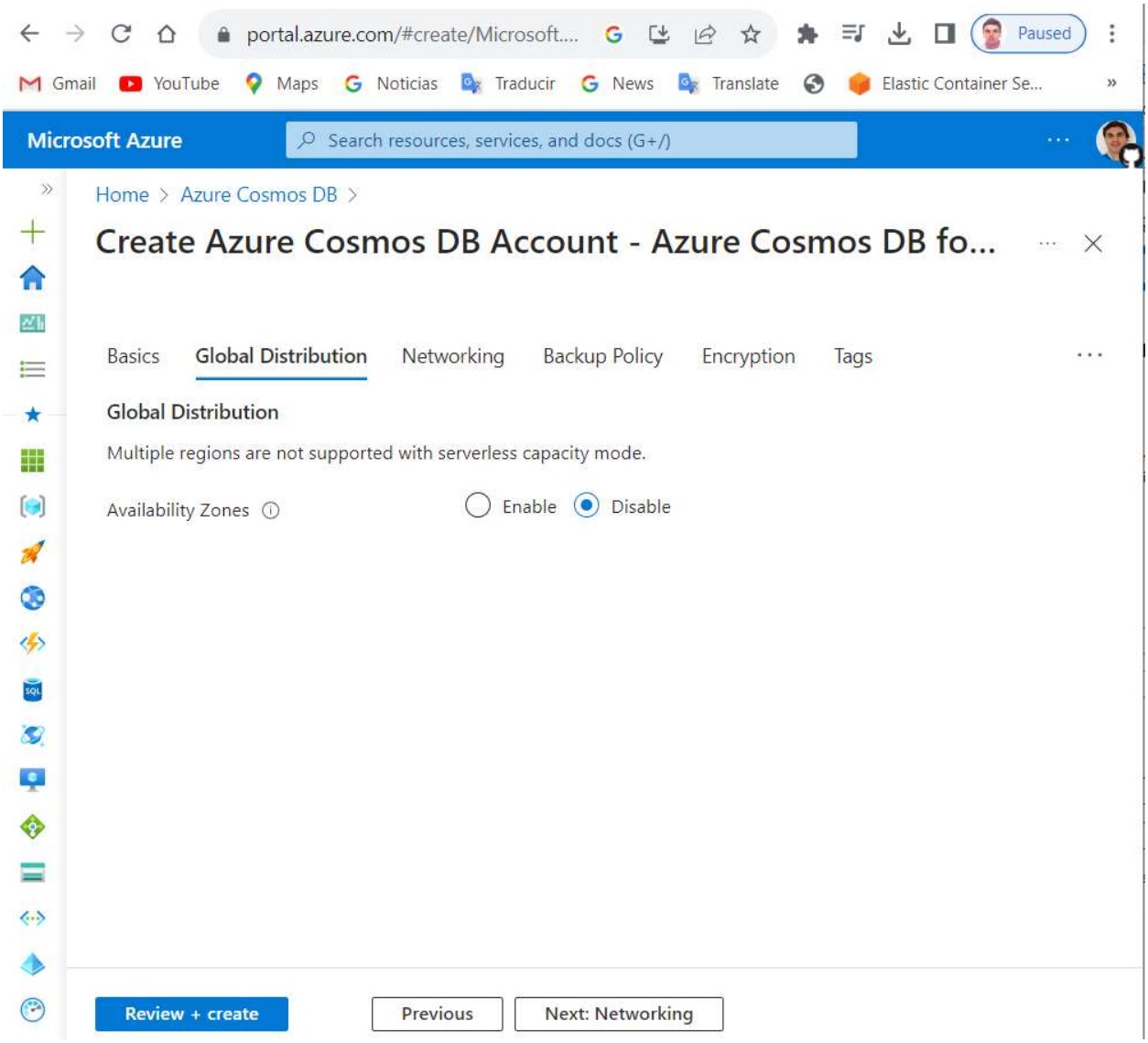
With Azure Cosmos DB free tier, you will get the first 1000 RU/s and 25 GB of storage for free in an account. You can enable free tier on up to one account per subscription. Estimated $64/month discount per account.

Apply Free Tier Discount Apply Do Not Apply

Limit total account throughput Limit the total amount of throughput that can be provisioned on this account

This limit will prevent unexpected charges related to provisioned throughput. You can update or remove this limit after your account is created.

Review + create Previous Next: Global Distribution


portal.azure.com/#create/Microsoft....
Paused
Gmail
YouTube
Maps
Noticias
Traducir
News
Translate
Elastic Container Se...
Microsoft Azure
Search resources, services, and docs (G+/)
Home > Azure Cosmos DB >
Create Azure Cosmos DB Account - Azure Cosmos DB fo...
Basics
Global Distribution
Networking
Backup Policy
Encryption
Tags
Global Distribution
Multiple regions are not supported with serverless capacity mode.
Availability Zones
Enable
Disable
Review + create
Previous
Next: Networking

portal.azure.com/#create/Microsoft....

Gmail YouTube Maps Noticias Traducir News Translate Elastic Container Se...

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB fo...

Basics | Global Distribution | Networking | Backup Policy | Encryption | Tags

Network connectivity

You can connect to your Azure Cosmos DB account either publically, via public IP addresses or service endpoints, or privately, using a private endpoint.

Connectivity method *

- (●) All networks
- ( ) Public endpoint (selected networks)
- ( ) Private endpoint

All networks will be able to access this CosmosDB account. Learn More

Connection Security Settings

Minimum Transport Layer Security Protocol ⓘ: TLS 1.2

Review + create | Previous | Next: Backup Policy

portal.azure.com/#create/Microsoft....

Gmail YouTube Maps Noticias Traducir News Translate Elastic Container Se...

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB fo...

Basics | Global Distribution | Networking | **Backup Policy** | Encryption | Tags

Azure Cosmos DB provides three different backup policies. You will not be able to switch to Periodic mode once you adopt Continuous mode. Learn more about the differences of the backup policies and pricing details.

Backup policy

- (●) Periodic
  Backup is taken at periodic interval based on your configuration
- ( ) Continuous (7 days)
  Provides backup window of 7 days / 168 hours and you can restore to any point of time within the window. This mode is available for free.
- ( ) Continuous (30 days)
  Provides backup window of 30 days / 720 hours and you can restore to any point of time within the window. This mode has cost impact.

Backup interval: 240 Minute(s)
60-1440

Backup retention: 8 Hours(s)
8-720

Copies of data retained: 2

Backup storage redundancy *

- (●) Geo-redundant backup storage
- ( ) Zone-redundant backup storage
- ( ) Locally-redundant backup storage

Review + create | Previous | Next: Encryption

portal.azure.com/#create/Microsoft.Docu...

Gmail YouTube Maps Noticias Traducir News Translate Elastic Container Se...

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB fo...

Basics | Global Distribution | Networking | Backup Policy | **Encryption** | Tags

Data Encryption

Azure Cosmos DB encryption protects your data at rest by seamlessly encrypting your data as it's written in our datacenters, and automatically decrypting it for you as you access it.

By default your Azure Cosmos DB account is encrypted at rest using service-managed keys. At the moment, you will not be able to switch back to service-managed key after opting into using custom-managed key while creating your account.
Learn More

Data Encryption

- (●) Service-managed key
- ( ) Customer-managed key (Enter key URI)

Review + create | Previous | Next: Tags

portal.azure.com/#create/Microsoft.Docu...

Gmail YouTube Maps Noticias Traducir News Translate Elastic Container Se...

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB fo...

Basics | Global Distribution | Networking | Backup Policy | Encryption | Tags

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. learn more

Note that if you create tags and then change resource settings on other tabs, your tags will be automatically updated.

Key	Value	Resource Type
		Azure Cosmos DB account

Review + create | Previous | Next: Review + create

portal.azure.com/#view/HubsExtension/DeploymentDetailsBlade/~/overview/id/%2Fsubscriptions%2F846901e6-da09-45c8-98ca-7cca2353ff0e%2FresourceGroups%2...

Gmail | YouTube | Maps | Noticias | Traducir | News | Translate | Elastic Container Se... | LattePanda 3 Delta... | Turkey launches TO... | ESP32 SIM800L GS... | Basic-ESP32-Tutori... | GPS Car Tracker Usi...

Microsoft Azure | Search resources, services, and docs (G+/) | luiscocoenriquez@hotm... PARTICULAR

Home >

Microsoft.Azure.CosmosDB-20231108103039 | Overview

Deployment

Search

- Overview
- Inputs
- Outputs
- Template

Delete | Cancel | Redeploy | Download | Refresh

Your deployment is complete

Deployment name : Microsoft.Azure.CosmosDB-20231108103039
Subscription : Azure subscription 1
Resource group : RG_CosmosDB

Start time : 11/8/2023, 10:30:40 AM
Correlation ID : f05c4949-81e8-4c9a-b4a8-8fee5f30a417

Deployment details

Next steps

Go to resource

Give feedback

Tell us about your experience with deployment

Cost management
Get notified to stay within your budget and prevent unexpected charges on your bill.
Set up cost alerts >

Microsoft Defender for Cloud
Secure your apps and infrastructure
Go to Microsoft Defender for Cloud >

Free Microsoft tutorials
Start learning today >

Work with an expert
Azure experts are service provider partners who can help manage your assets on Azure and be your first line of support.
Find an Azure expert >

portal.azure.com/#@luiscocoenriquezhotmail.onmicrosoft.com/resource/subscriptions/846901e6-da09-45c8-98ca-7cca2353

Gmail YouTube Maps Noticias Traducir News Translate Elastic Container Se... LattePanda 3 Delta... Turk

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Microsoft.Azure.CosmosDB-20231108103039 | Overview > luiscosmosdb

luiscosmosdb | Quick start

Azure Cosmos DB account

Search

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Quick start
- Notifications
- Data Explorer

Settings

- Features
- Default consistency
- Backup & Restore
- Networking
- CORS
- Dedicated Gateway
- Keys

Congratulations! Your Azure Cosmos DB account was created.

Now, let's connect to it using a sample app:

Choose a platform

.NET Java Node.js Python

1 Step 1: Add a container

In Azure Cosmos DB, data is stored in containers.

Create 'Items' container

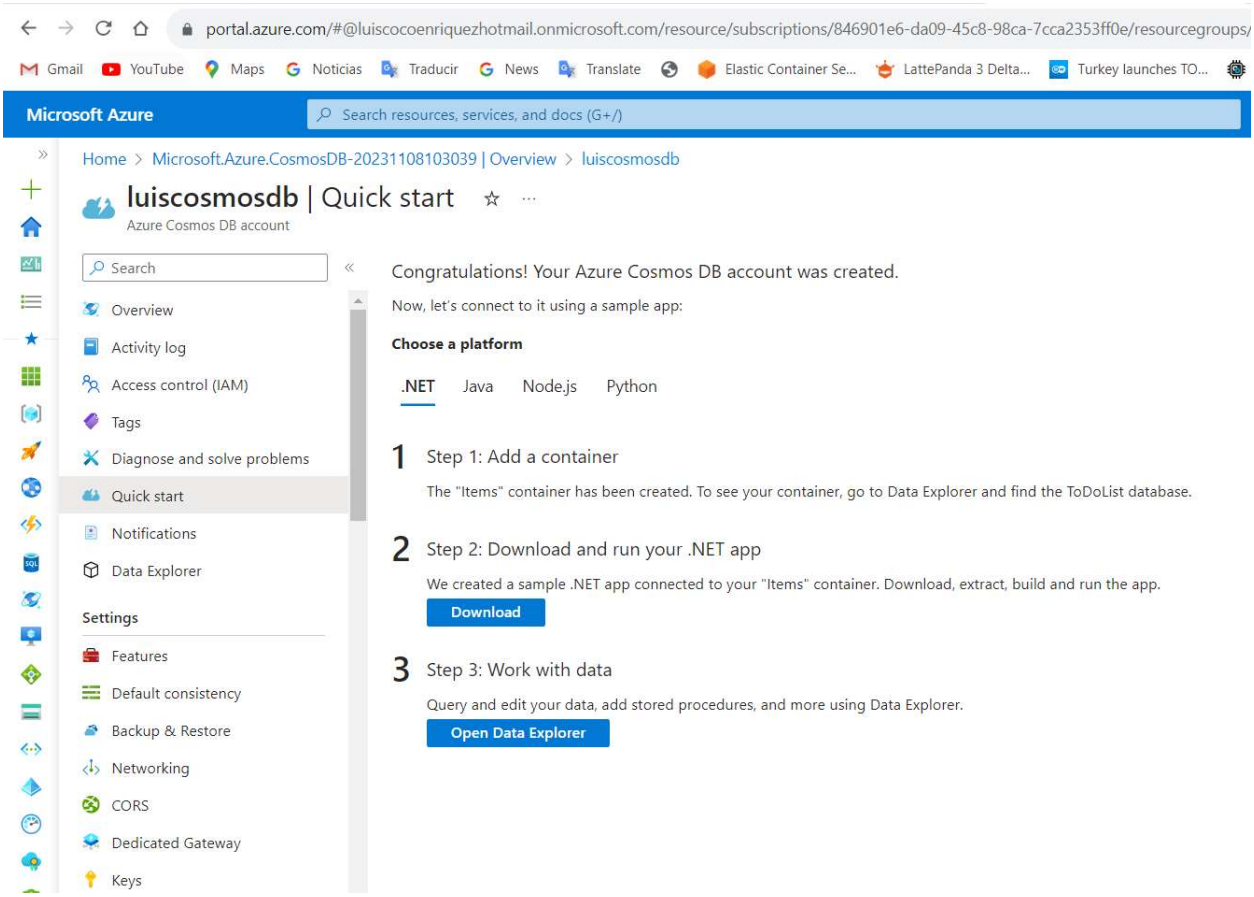
Create 'Items' container. To see your container, go to Data Explorer and find the ToDoList database.

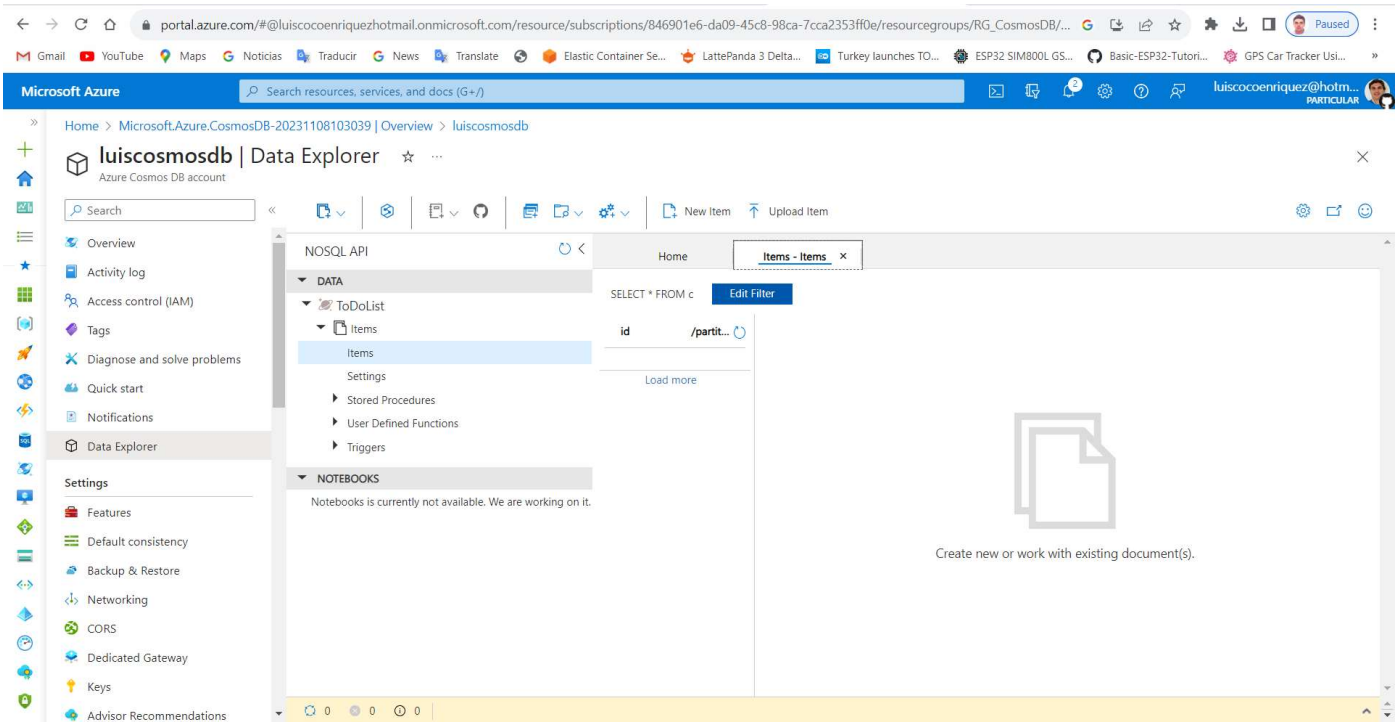
2 Step 2: Download and run your .NET app

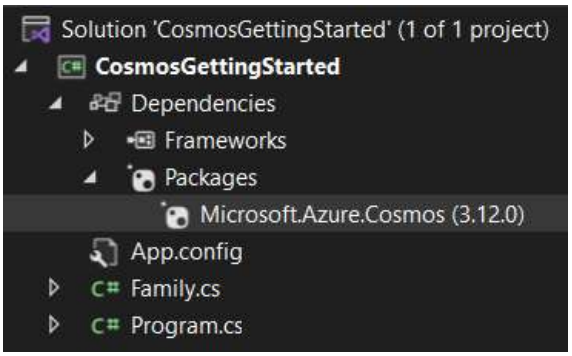
Once container is created, download a sample .NET app connected to it, extract, build and run.

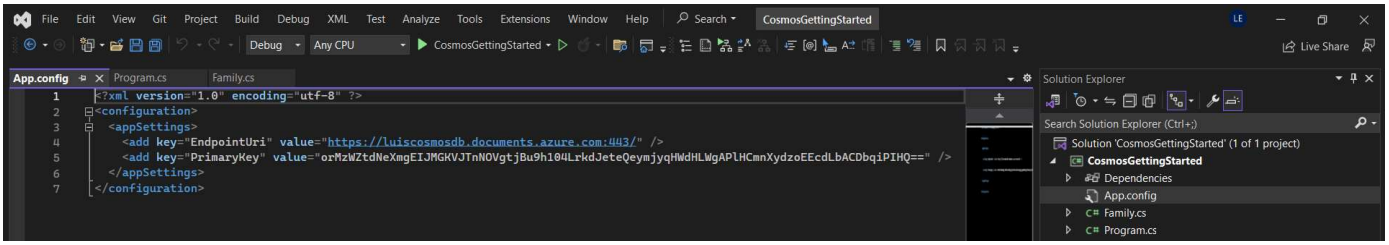
Download

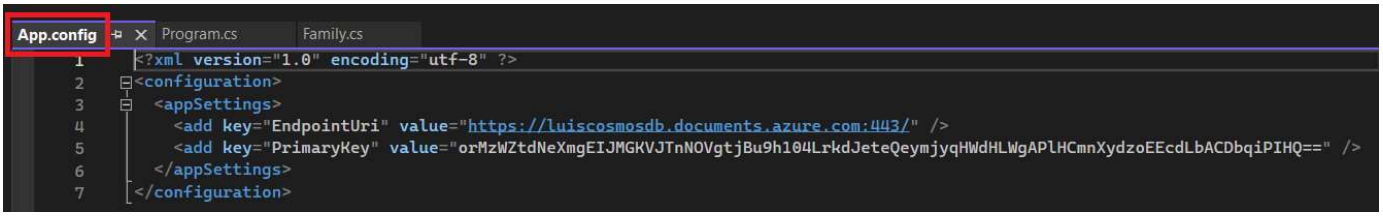
.NET Sample

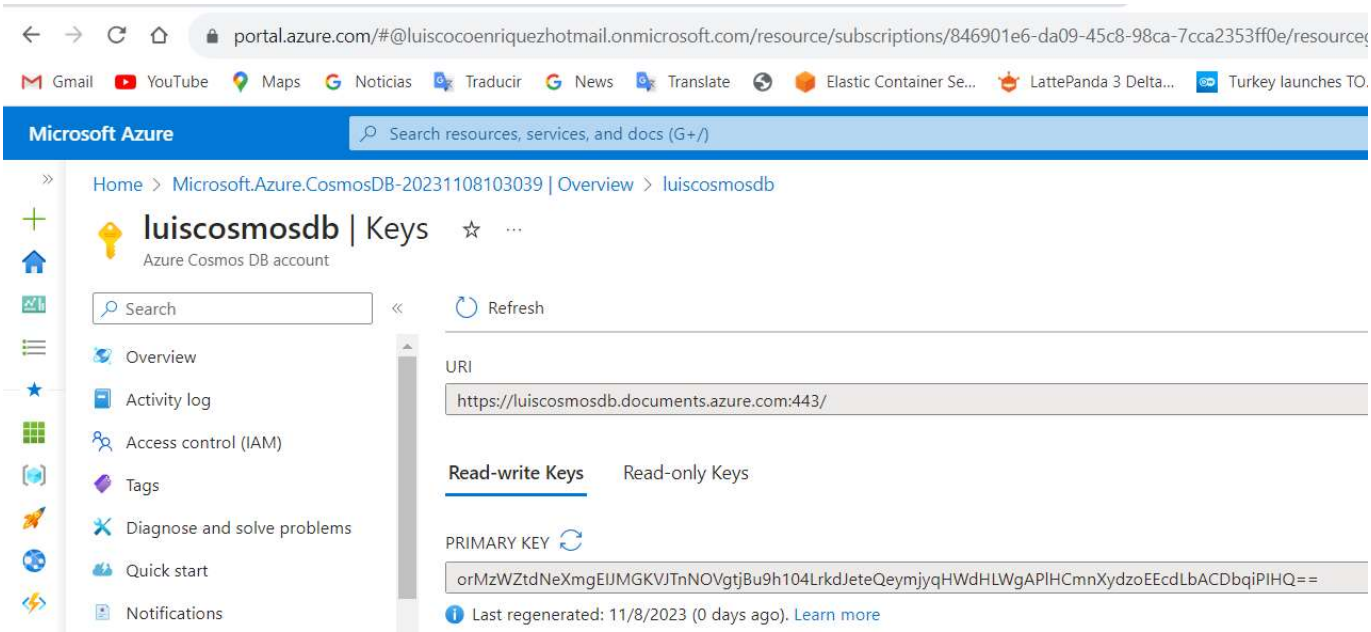

portal.azure.com/#@luiscocoenriquezhotmail.onmicrosoft.com/resource/subscriptions/846901e6-da09-45c8-98ca-7cca2353ff0e/resourcegroups/
Gmail
YouTube
Maps
Noticias
Traducir
News
Translate
Elastic Container Se...
LattePanda 3 Delta...
Turkey launches TO...
Microsoft Azure
Search resources, services, and docs (G+/)
Home > Microsoft.Azure.CosmosDB-20231108103039 | Overview > luiscosmosdb
luiscosmosdb | Quick start
Azure Cosmos DB account
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Quick start
Notifications
Data Explorer
Settings
Features
Default consistency
Backup & Restore
Networking
CORS
Dedicated Gateway
Keys
Congratulations! Your Azure Cosmos DB account was created.
Now, let's connect to it using a sample app:
Choose a platform
.NET
Java
Node.js
Python
1 Step 1: Add a container
The "Items" container has been created. To see your container, go to Data Explorer and find the ToDoList database.
2 Step 2: Download and run your .NET app
We created a sample .NET app connected to your "Items" container. Download, extract, build and run the app.
Download
3 Step 3: Work with data
Query and edit your data, add stored procedures, and more using Data Explorer.
Open Data Explorer


portal.azure.com/#@luiscocoenriquezhotmail.onmicrosoft.com/resource/subscriptions/846901e6-da09-45c8-98ca-7cca2353ff0e/resourcegroups/RG_CosmosDB/...
Microsoft Azure
Search resources, services, and docs (G+/)
luiscocoenriquez@hotm...
PARTICULAR
Home > Microsoft.Azure.CosmosDB-20231108103039 | Overview > luiscosmosdb
luiscosmosdb | Data Explorer
Azure Cosmos DB account
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Quick start
Notifications
Data Explorer
Settings
Features
Default consistency
Backup & Restore
Networking
CORS
Dedicated Gateway
Keys
Advisor Recommendations
New Item
Upload Item
NOSQL API
DATA
ToDoList
Items
Items
Settings
Stored Procedures
User Defined Functions
Triggers
NOTEBOOKS
Notebooks is currently not available. We are working on it.
Home
Items - Items
SELECT * FROM c
Edit Filter
id
/partit...
Load more
Create new or work with existing document(s).

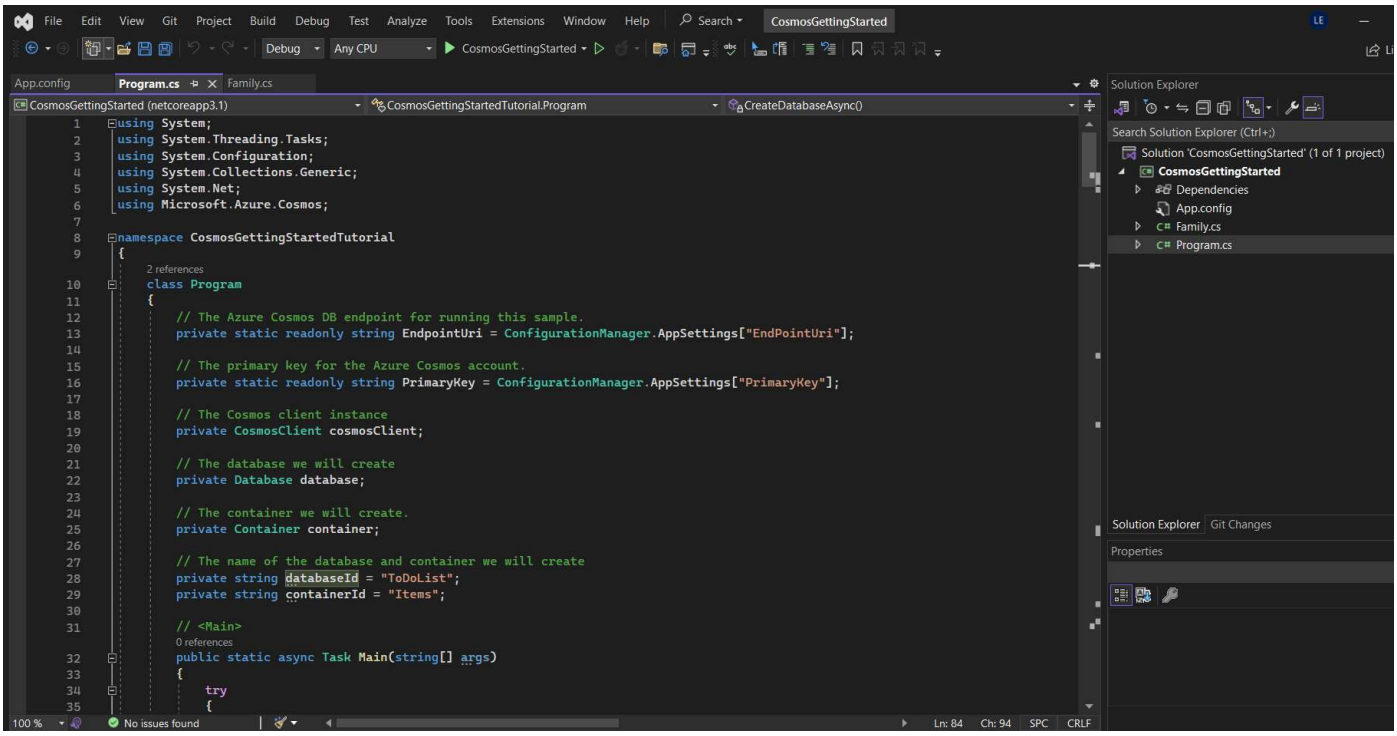

Solution 'CosmosGettingStarted' (1 of 1 project)
CosmosGettingStarted
Dependencies
Frameworks
Packages
Microsoft.Azure.Cosmos (3.12.0)
App.config
Family.cs
Program.cs


File Edit View Git Project Build Debug XML Test Analyze Tools Extensions Window Help
Search
CosmosGettingStarted
Debug
Any CPU
CosmosGettingStarted
Live Share
App.config
Program.cs
Family.cs
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<appSettings>
<add key="EndpointUri" value="https://luiscosmosdb.documents.azure.com:443/" />
<add key="PrimaryKey" value="orMzWZtdNeXmgEIJMGKVJTnNOVgtjBu9h104LrkdJeteQeymjyqHWdHLWgAPlHCmnXydzoEEcdLbACDbqiPIHQ==" />
</appSettings>
</configuration>
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'CosmosGettingStarted' (1 of 1 project)
CosmosGettingStarted
Dependencies
App.config
Family.cs
Program.cs


App.config
Program.cs
Family.cs
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<appSettings>
<add key="EndpointUri" value="https://luiscosmosdb.documents.azure.com:443/" />
<add key="PrimaryKey" value="orMzWZtdNeXmgEIJMGKVJTnNOVgtjBu9h104LrkdJeteQeymjyqHWdHLWgAPlHCmnXydzoEEcdLbACDbqiPIHQ==" />
</appSettings>
</configuration>



Now we run the .Net sample application



Microsoft Visual Studio Debug Console

```
You must install or update .NET to run this application.

App: C:\Users\LEnriquez\3D Objects\Downloads\DocumentDB-Quickstart-DotNet\sql-dotnet\CosmosGettingStartedTutorial\bin\De
bug\netcoreapp3.1\CosmosGettingStarted.exe
Architecture: x64
Framework: 'Microsoft.NETCore.App', version '3.1.0' (x64)
.NET location: C:\Program Files\dotnet\

The following frameworks were found:
  2.0.0 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  2.2.0 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  2.2.8 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  6.0.10 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  6.0.23 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  6.0.24 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  7.0.12 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  7.0.13 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]

Learn about framework resolution:
https://aka.ms/dotnet/app-launch-failed

To install missing framework, download:
https://aka.ms/dotnet-core-applaunch?framework=Microsoft.NETCore.App&framework_version=3.1.0&arch=x64&rid=win10-x64

C:\Users\LEnriquez\3D Objects\Downloads\DocumentDB-Quickstart-DotNet\sql-dotnet\CosmosGettingStartedTutorial\bin\Debug\n
etcoreapp3.1\CosmosGettingStarted.exe (process 4384) exited with code -2147450730.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

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));
```

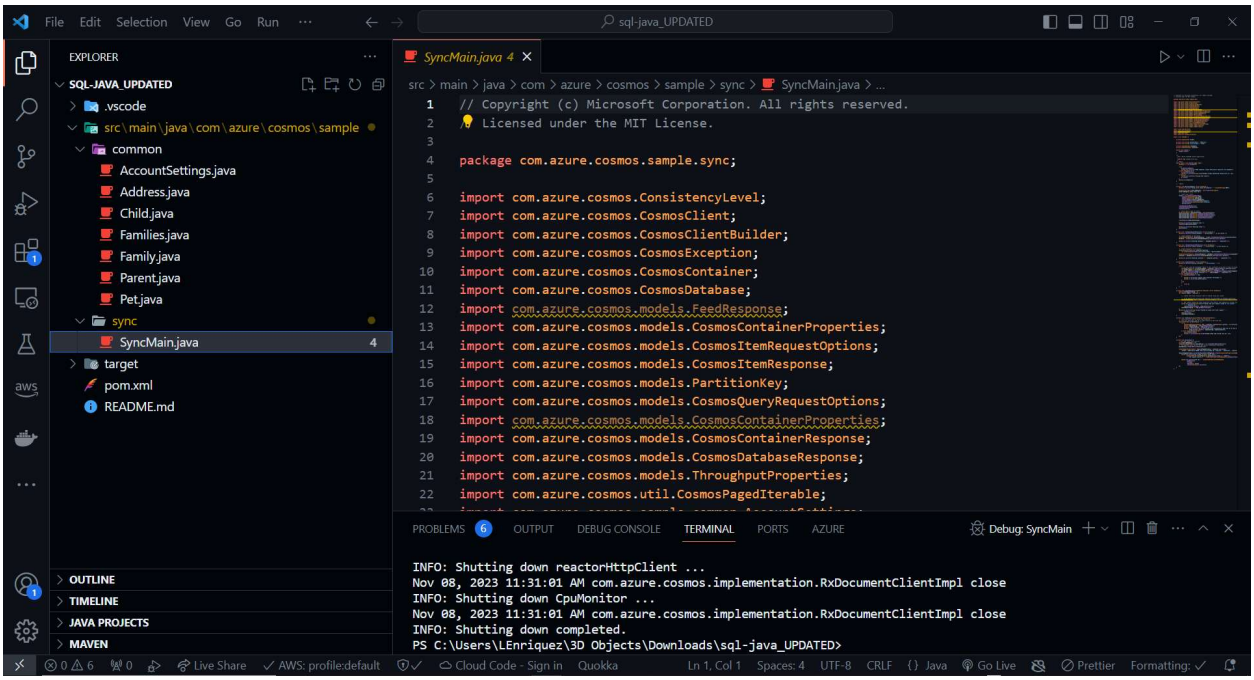
```
        Console.WriteLine("Updated Family [{0},{1}].\n \tBody is now: {2}\n", itemBody.LastName, itemBody.Id, wakefieldFamilyResponse.Resource);
    }
    // </ReplaceFamilyItemAsync>

    // <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



pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.azure</groupId>
    <artifactId>azure-cosmos-java-sql-api-samples</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>Get Started With Sync / Async Java SDK for SQL API of Azure Cosmos DB
Database Service
    </name>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>


    <build>
        <plugins>
```

```
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.6.0</version>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-eclipse-plugin</artifactId>
                <version>2.8</version>
                <configuration>
                    <classpathContainers>
                        <classpathContainer>
                            org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.j
dt.internal.debug.ui.launcher.StandardVMType/JavaSE-1.8
                        </classpathContainer>
                    </classpathContainers>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>com.azure</groupId>
            <artifactId>azure-cosmos</artifactId>
            <version>4.43.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-slf4j-impl</artifactId>
            <version>2.13.0</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.11.1</version>
```

```
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-jdk14</artifactId>
            <version>1.7.28</version>
        </dependency>

        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.10</version>
        </dependency>
    </dependencies>
</project>
```

AccountSettings.java

```
package com.azure.cosmos.sample.common;

import org.apache.commons.lang3.StringUtils;

public class AccountSettings {
    // Replace MASTER_KEY and HOST with values from your Azure Cosmos DB account.
    // The default values are credentials of the local emulator, which are not
used in any production environment.
    // <!--[SuppressMessage("Microsoft.Security", "CS002:SecretInNextLine")]-->
    public static String MASTER_KEY =
            System.getProperty("ACCOUNT_KEY",
                    StringUtils.defaultString(StringUtils.trimToNull(
                            System.getenv().get("ACCOUNT_KEY")),
                            "orMzWZtdNeXmgEIJMGKVJTnNOVgtjBu9h104LrkdJeteQeymjyqH
WdHLWgAPlHCmnXydzoEEcdLbACDbqiPIHQ==")));

    public static String HOST =
            System.getProperty("ACCOUNT_HOST",
                    StringUtils.defaultString(StringUtils.trimToNull(
                            System.getenv().get("ACCOUNT_HOST")),
                            "https://luiscosmosdb.documents.azure.com:443/"));
}
```

SyncMain.java

```java
// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.

package com.azure.cosmos.sample.sync;

import com.azure.cosmos.ConsistencyLevel;
import com.azure.cosmos.CosmosClient;
import com.azure.cosmos.CosmosClientBuilder;
import com.azure.cosmos.CosmosException;
import com.azure.cosmos.CosmosContainer;
import com.azure.cosmos.CosmosDatabase;
import com.azure.cosmos.models.FeedResponse;
import com.azure.cosmos.models.CosmosContainerProperties;
import com.azure.cosmos.models.CosmosItemRequestOptions;
import com.azure.cosmos.models.CosmosItemResponse;
import com.azure.cosmos.models.PartitionKey;
import com.azure.cosmos.models.CosmosQueryRequestOptions;
import com.azure.cosmos.models.CosmosContainerProperties;
import com.azure.cosmos.models.CosmosContainerResponse;
import com.azure.cosmos.models.CosmosDatabaseResponse;
import com.azure.cosmos.models.ThroughputProperties;
import com.azure.cosmos.util.CosmosPagedIterable;
import com.azure.cosmos.sample.common.AccountSettings;
import com.azure.cosmos.sample.common.Families;
import com.azure.cosmos.sample.common.Family;

import java.time.Duration;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.stream.Collectors;

public class SyncMain {

    private CosmosClient client;

    private final String databaseName = "ToDoList";
    private final String containerName = "Items";

    private CosmosDatabase database;
    private CosmosContainer container;

    public void close() {
```

```
        client.close();
    }

    /**
     * Run a Hello CosmosDB console application.
     *
     * @param args command line args.
     */
    //  <Main>
    public static void main(String[] args) {
        SyncMain p = new SyncMain();

        try {
            p.getStartedDemo();
            System.out.println("Demo complete, please hold while resources are
released");
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println(String.format("Cosmos getStarted failed with %s",
e));
        } finally {
            System.out.println("Closing the client");
            p.close();
        }
        System.exit(0);
    }

    //  </Main>

    private void getStartedDemo() throws Exception {
        System.out.println("Using Azure Cosmos DB endpoint: " +
AccountSettings.HOST);

        ArrayList<String> preferredRegions = new ArrayList<String>();
        preferredRegions.add("West US");

        //  Create sync client
        client = new CosmosClientBuilder()
            .endpoint(AccountSettings.HOST)
            .key(AccountSettings.MASTER_KEY)
            .preferredRegions(preferredRegions)
            .userAgentSuffix("CosmosDBJavaQuickstart")
            .consistencyLevel(ConsistencyLevel.EVENTUAL)
            .buildClient();
```

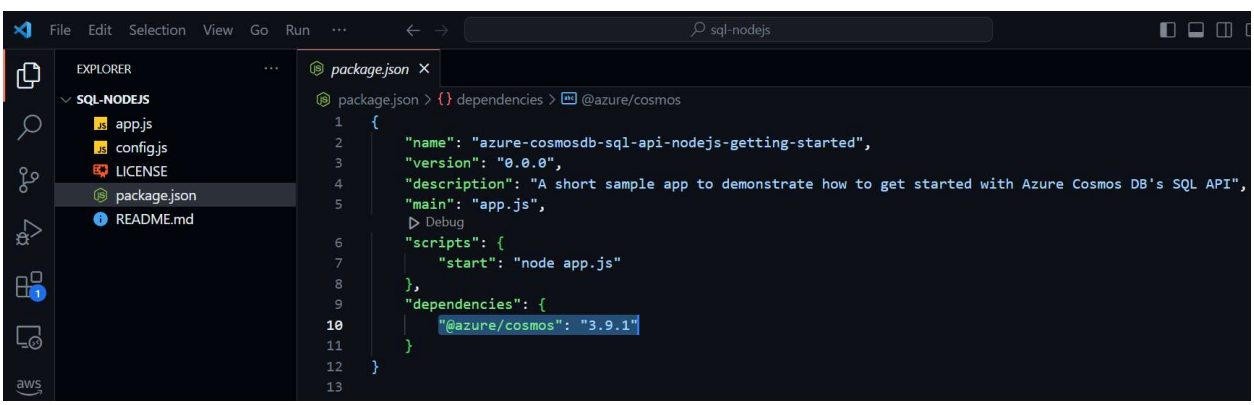
```
        createDatabaseIfNotExists();
        createContainerIfNotExists();
        scaleContainer();

        //  Setup family items to create
        ArrayList<Family> familiesToCreate = new ArrayList<>();
        familiesToCreate.add(Families.getAndersenFamilyItem());
        familiesToCreate.add(Families.getWakefieldFamilyItem());
        familiesToCreate.add(Families.getJohnsonFamilyItem());
        familiesToCreate.add(Families.getSmithFamilyItem());

        createFamilies(familiesToCreate);

        System.out.println("Reading items.");
        readItems(familiesToCreate);

        System.out.println("Querying items.");
        queryItems();
    }

    private void createDatabaseIfNotExists() throws Exception {
        System.out.println("Create database " + databaseName + " if not
exists.");

        //  Create database if not exists
        CosmosDatabaseResponse databaseResponse =
client.createDatabaseIfNotExists(databaseName);
        database = client.getDatabase(databaseResponse.getProperties().getId());

        System.out.println("Checking database " + database.getId() + "
completed!\n");
    }

    private void createContainerIfNotExists() throws Exception {
        System.out.println("Create container " + containerName + " if not
exists.");

        //  Create container if not exists
        CosmosContainerProperties containerProperties =
            new CosmosContainerProperties(containerName, "/partitionKey");

        CosmosContainerResponse containerResponse =
database.createContainerIfNotExists(containerProperties);
        container =
database.getContainer(containerResponse.getProperties().getId());
```

```
        System.out.println("Checking container " + container.getId() + "
completed!\n");
    }

    private void scaleContainer() throws Exception {
        System.out.println("Scaling container " + containerName + ".");

        try {
            // 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
            ThroughputProperties currentThroughput =
container.readThroughput().getProperties();
            int newThroughput = currentThroughput.getManualThroughput() + 100;
            container.replaceThroughput(ThroughputProperties.createManualThroughp
ut(newThroughput));
            System.out.println("Scaled container to " + newThroughput + "
completed!\n");
        } catch (CosmosException e) {
            if (e.getStatusCode() == 400)
            {
                System.err.println("Cannot read container throuthput.");
                System.err.println(e.getMessage());
            }
            else
            {
                throw e;
            }
        }
    }

    private void createFamilies(List<Family> families) throws Exception {
        double totalRequestCharge = 0;
        for (Family family : families) {

            //  Create item using container that we created using sync client

            //  Using appropriate partition key improves the performance of
database operations
            CosmosItemResponse item = container.createItem(family, new
PartitionKey(family.getPartitionKey()), new CosmosItemRequestOptions());

            //  Get request charge and other properties like latency, and
diagnostics strings, etc.
```

```
            System.out.println(String.format("Created item with request charge of
%.2f within" +
                    " duration %s",
                item.getRequestCharge(), item.getDuration()));
            totalRequestCharge += item.getRequestCharge();
        }
        System.out.println(String.format("Created %d items with total request " +
                "charge of %.2f",
            families.size(),
            totalRequestCharge));
    }

    private void readItems(ArrayList<Family> familiesToCreate) {
        //  Using partition key for point read scenarios.
        //  This will help fast look up of items because of partition key
        familiesToCreate.forEach(family -> {
            try {
                CosmosItemResponse<Family> item =
container.readItem(family.getId(), new PartitionKey(family.getPartitionKey()),
Family.class);
                double requestCharge = item.getRequestCharge();
                Duration requestLatency = item.getDuration();
                System.out.println(String.format("Item successfully read with id
%s with a charge of %.2f and within duration %s",
                        item.getItem().getId(), requestCharge, requestLatency));
            } catch (CosmosException e) {
                e.printStackTrace();
                System.err.println(String.format("Read Item failed with %s", e));
            }
        });
    }

    private void queryItems() {
        // Set some common query options
        int preferredPageSize = 10;
        CosmosQueryRequestOptions queryOptions = new CosmosQueryRequestOptions();
        //  Set populate query metrics to get metrics around query executions
        queryOptions.setQueryMetricsEnabled(true);

        CosmosPagedIterable<Family> familiesPagedIterable = container.queryItems(
            "SELECT * FROM Family WHERE Family.partitionKey IN ('Andersen',
'Wakefield', 'Johnson')", queryOptions, Family.class);

        familiesPagedIterable.iterableByPage(preferredPageSize).forEach(cosmosIte
mPropertiesFeedResponse -> {
```

```
            System.out.println("Got a page of query result with " +
                    cosmosItemPropertiesFeedResponse.getResults().size() + "
items(s)"
                    + " and request charge of " +
cosmosItemPropertiesFeedResponse.getRequestCharge());

            System.out.println("Item Ids " + cosmosItemPropertiesFeedResponse
                    .getResults()
                    .stream()
                    .map(Family::getId)
                    .collect(Collectors.toList()));
        });
    }
}
```

Node.js Sample



package.json

```
{
    "name": "azure-cosmosdb-sql-api-nodejs-getting-started",
    "version": "0.0.0",
    "description": "A short sample app to demonstrate how to get started with
Azure Cosmos DB's SQL API",
    "main": "app.js",
    "scripts": {
        "start": "node app.js"
    },
    "dependencies": {
        "@azure/cosmos": "3.9.1"
```

```
    }
}
```

config.js

```
var config = {}

config.endpoint = 'https://luiscosmosdb.documents.azure.com:443/'
config.key =
'orMzWZtdNeXmgEIJMGKVJTnNOVgtjBu9h104LrkdJeteQeymjyqHWdHLWgAPlHCmnXydzoEEcdLbACDb
qiPIHQ=='

config.database = {
  id: 'ToDoList'
}

config.container = {
  id: 'Items'
}

config.items = {
  Andersen: {
    id: 'Anderson.1',
    Country: 'USA',
    partitionKey: 'USA',
    lastName: 'Andersen',
    parents: [
      {
        firstName: 'Thomas'
      },
      {
        firstName: 'Mary Kay'
      }
    ],
    children: [
      {
        firstName: 'Henriette Thaulow',
        gender: 'female',
        grade: 5,
        pets: [
          {
            givenName: 'Fluffy'
          }
        ]
```

```
      }
    ],
    address: {
      state: 'WA',
      county: 'King',
      city: 'Seattle'
    }
  },
  Wakefield: {
    id: 'Wakefield.7',
    partitionKey: 'Italy',
    Country: 'Italy',
    parents: [
      {
        familyName: 'Wakefield',
        firstName: 'Robin'
      },
      {
        familyName: 'Miller',
        firstName: 'Ben'
      }
    ],
    children: [
      {
        familyName: 'Merriam',
        firstName: 'Jesse',
        gender: 'female',
        grade: 8,
        pets: [
          {
            givenName: 'Goofy'
          },
          {
            givenName: 'Shadow'
          }
        ]
      },
      {
        familyName: 'Miller',
        firstName: 'Lisa',
        gender: 'female',
        grade: 1
      }
    ],
    address: {
```

```
      state: 'NY',
      county: 'Manhattan',
      city: 'NY'
    },
    isRegistered: false
  }
}

module.exports = config
```

app.js

```
//@ts-check
const CosmosClient = require('@azure/cosmos').CosmosClient

const config = require('./config')
const url = require('url')

const endpoint = config.endpoint
const key = config.key

const databaseId = config.database.id
const containerId = config.container.id
const partitionKey = { kind: 'Hash', paths: ['/partitionKey'] }

const options = {
      endpoint: endpoint,
      key: key,
      userAgentSuffix: 'CosmosDBJavascriptQuickstart'
    };

const client = new CosmosClient(options)

/**
 * Create the database if it does not exist
 */
async function createDatabase() {
  const { database } = await client.databases.createIfNotExists({
    id: databaseId
  })
  console.log(`Created database:\n${database.id}\n`)
}

/**
```

```
 * Read the database definition
 */
async function readDatabase() {
  const { resource: databaseDefinition } = await client
    .database(databaseId)
    .read()
  console.log(`Reading database:\n${databaseDefinition.id}\n`)
}

/**
 * Create the container if it does not exist
 */
async function createContainer() {
  const { container } = await client
    .database(databaseId)
    .containers.createIfNotExists(
      { id: containerId, partitionKey }
    )
  console.log(`Created container:\n${config.container.id}\n`)
}

/**
 * Read the container definition
 */
async function readContainer() {
  const { resource: containerDefinition } = await client
    .database(databaseId)
    .container(containerId)
    .read()
  console.log(`Reading container:\n${containerDefinition.id}\n`)
}

/**
 * Scale a 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
 */
async function scaleContainer() {
  const { resource: containerDefinition } = await client
    .database(databaseId)
    .container(containerId)
    .read();

  try
  {
```

```
      const {resources: offers} = await client.offers.readAll().fetchAll();

      const newRups = 500;
      for (var offer of offers) {
        if (containerDefinition._rid !== offer.offerResourceId)
        {
            continue;
        }
        offer.content.offerThroughput = newRups;
        const offerToReplace = client.offer(offer.id);
        await offerToReplace.replace(offer);
        console.log(`Updated offer to ${newRups} RU/s\n`);
        break;
      }
  }
  catch(err)
  {
      if (err.code == 400)
      {
          console.log(`Cannot read container throuthput.\n`);
          console.log(err.body.message);
      }
      else
      {
          throw err;
      }
  }
}

/**
 * Create family item if it does not exist
 */
async function createFamilyItem(itemBody) {
  const { item } = await client
    .database(databaseId)
    .container(containerId)
    .items.upsert(itemBody)
  console.log(`Created family item with id:\n${itemBody.id}\n`)
}

/**
 * Query the container using SQL
 */
async function queryContainer() {
  console.log(`Querying container:\n${config.container.id}`)
```

```
  // query to return all children in a family
  // Including the partition key value of country in the WHERE filter results in
a more efficient query
  const querySpec = {
    query: 'SELECT VALUE r.children FROM root r WHERE r.partitionKey = @country',
    parameters: [
      {
        name: '@country',
        value: 'USA'
      }
    ]
  }

  const { resources: results } = await client
    .database(databaseId)
    .container(containerId)
    .items.query(querySpec)
    .fetchAll()
  for (var queryResult of results) {
    let resultString = JSON.stringify(queryResult)
    console.log(`\tQuery returned ${resultString}\n`)
  }
}

/**
 * Replace the item by ID.
 */
async function replaceFamilyItem(itemBody) {
  console.log(`Replacing item:\n${itemBody.id}\n`)
  // Change property 'grade'
  itemBody.children[0].grade = 6
  const { item } = await client
    .database(databaseId)
    .container(containerId)
    .item(itemBody.id, itemBody.partitionKey)
    .replace(itemBody)
}

/**
 * Delete the item by ID.
 */
async function deleteFamilyItem(itemBody) {
  await client
    .database(databaseId)
```

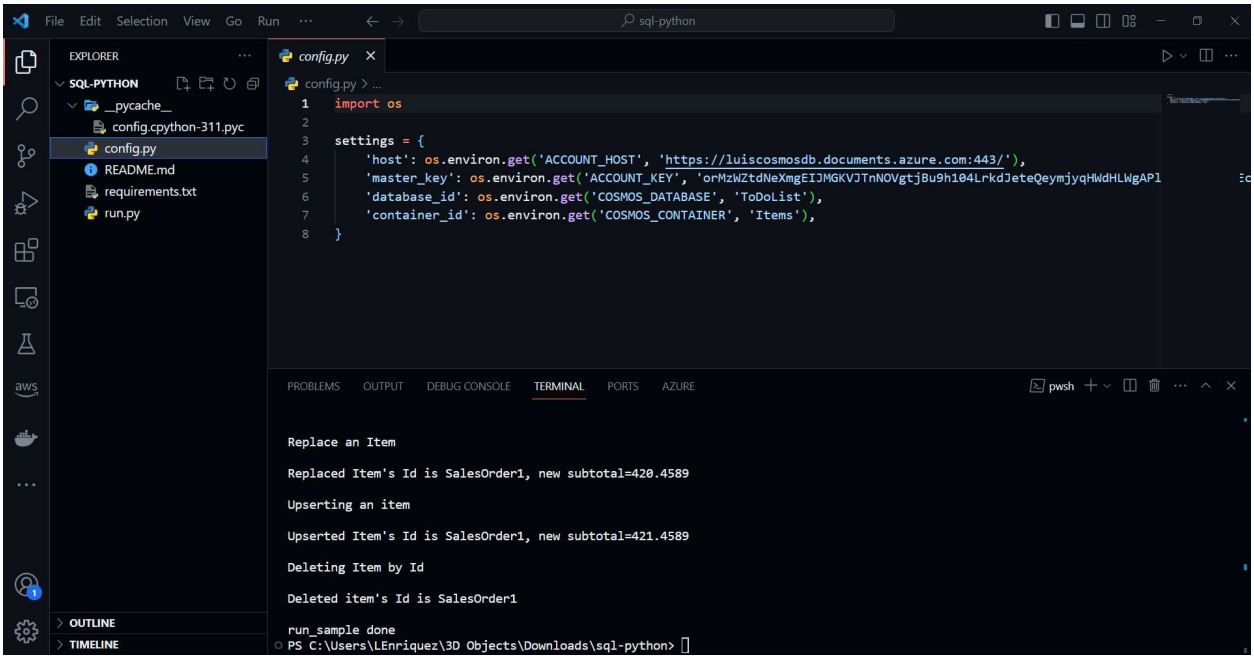
```
    .container(containerId)
    .item(itemBody.id, itemBody.partitionKey)
    .delete(itemBody)
  console.log(`Deleted item:\n${itemBody.id}\n`)
}

/**
 * Cleanup the database and collection on completion
 */
async function cleanup() {
  await client.database(databaseId).delete()
}

/**
 * Exit the app with a prompt
 * @param {string} message - The message to display
 */
function exit(message) {
  console.log(message)
  console.log('Press any key to exit')
  //process.stdin.setRawMode(true)
  process.stdin.resume()
  process.stdin.on('data', process.exit.bind(process, 0))
}

createDatabase()
  .then(() => readDatabase())
  .then(() => createContainer())
  .then(() => readContainer())
  .then(() => scaleContainer())
  .then(() => createFamilyItem(config.items.Andersen))
  .then(() => createFamilyItem(config.items.Wakefield))
  .then(() => queryContainer())
  .then(() => replaceFamilyItem(config.items.Andersen))
  .then(() => queryContainer())
  .then(() => deleteFamilyItem(config.items.Andersen))
  .then(() => {
    exit(`Completed successfully`)
  })
  .catch(error => {
    exit(`Completed with error ${JSON.stringify(error)}`)
  })
```

To install the application dependencies:

npm i

And then to run the application in VS Code

Python Sample



---
page_type: sample
languages:
- python
products:
- azure
description: "Azure Cosmos DB is a globally distributed multi-model database."

Developing a Python app using the Azure Cosmos DB SQL API

Azure Cosmos DB is a globally distributed multi-model database. One of the supported APIs is the SQL API, which provides a JSON document model with SQL

querying and JavaScript procedural logic. This sample shows you how to use Azure Cosmos DB with the SQL API to store and access data from a Node.js application.

Running this sample

* Before you can run this sample, you must have the following perquisites:
    * An active Azure Cosmos DB account - If you don't have an account, refer to the [Create an Azure Cosmos DB account](https://learn.microsoft.com/azure/cosmos-db/nosql/quickstart-python?tabs=azure-portal%2Cpasswordless%2Cwindows%2Csign-in-azure-cli%2Csync) article.
    * [Python](https://www.python.org/downloads/) version v3.7 or higher.
    * [Git](http://git-scm.com/).

1. We created a sample Python app connected to your "Items" collection. Download and extract the app from the Azure Cosmos DB portal.

2. Change directories to the repo using `cd sql-python` or open with your favorite IDE (Visual Studio/Visual Studio Code).

3. There is no need to change the credentials from the extracted app. We have entered the Azure Cosmos DB endpoint URL and the key into your `config.py` file.

4. Run `pip install -r requirements.txt` in a terminal to install required python packages.

5. Run `python run.py` in a terminal to start your start your node application.

About the code

The code included in this sample is intended to get you quickly started with a Node.js console application that connects to Azure Cosmos DB with the SQL API.

More information

- [Azure Cosmos DB](https://docs.microsoft.com/azure/cosmos-db/introduction)

- [Azure Cosmos DB: SQL API](https://docs.microsoft.com/en-us/azure/cosmos-db/sql-api-introduction)
- [Azure Cosmos DB Node.js SDK](https://docs.microsoft.com/en-us/azure/cosmos-db/sql-api-sdk-node)
- [Azure Cosmos DB Node.js SDK Reference Documentation](http://azure.github.io/azure-documentdb-node/)

config.py

```
import os

settings = {
    'host': os.environ.get('ACCOUNT_HOST',
'https://luiscosmosdb.documents.azure.com:443/'),
    'master_key': os.environ.get('ACCOUNT_KEY',
'orMzWZtdNeXmgEIJMGKVJTnNOVgtjBu9h104LrkdJeteQeymjyqHWdHLWgAPlHCmnXydzoEEcdLbACDb
qiPIHQ=='),
    'database_id': os.environ.get('COSMOS_DATABASE', 'ToDoList'),
    'container_id': os.environ.get('COSMOS_CONTAINER', 'Items'),
}
```

run.py

```
import azure.cosmos.documents as documents
import azure.cosmos.cosmos_client as cosmos_client
import azure.cosmos.exceptions as exceptions
from azure.cosmos.partition_key import PartitionKey
import datetime

import config

# ------------------------------------------------------------------------------
--------------------------
# Prerequistes -
#
# 1. An Azure Cosmos account -
#    https://docs.microsoft.com/azure/cosmos-db/create-cosmosdb-resources-
portal#create-an-azure-cosmos-db-account
#
# 2. Microsoft Azure Cosmos PyPi package -
#    https://pypi.python.org/pypi/azure-cosmos/
```

```
# ----------------------------------------------------------------------------
---------------------------
# Sample - demonstrates the basic CRUD operations on a Item resource for Azure
Cosmos
# ----------------------------------------------------------------------------
---------------------------

HOST = config.settings['host']
MASTER_KEY = config.settings['master_key']
DATABASE_ID = config.settings['database_id']
CONTAINER_ID = config.settings['container_id']


def create_items(container):
    print('\nCreating Items\n')

    # Create a SalesOrder object. This object has nested properties and various
types including numbers, DateTimes and strings.
    # This can be saved as JSON as is without converting into rows/columns.
    sales_order = get_sales_order("SalesOrder1")
    container.create_item(body=sales_order)

    # As your app evolves, let's say your object has a new schema. You can insert
SalesOrderV2 objects without any
    # changes to the database tier.
    sales_order2 = get_sales_order_v2("SalesOrder2")
    container.create_item(body=sales_order2)

def scale_container(container):
    print('\nScaling Container\n')

    # 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
    try:
        offer = container.read_offer()
        print('Found Offer and its throughput is
\'{0}\''.format(offer.offer_throughput))

        offer.offer_throughput += 100
        container.replace_throughput(offer.offer_throughput)

        print('Replaced Offer. Offer Throughput is now
\'{0}\''.format(offer.offer_throughput))

    except exceptions.CosmosHttpResponseError as e:
```

```
        if e.status_code == 400:
            print('Cannot read container throuthput.');
            print(e.http_error_message);
        else:
            raise;

def read_item(container, doc_id, account_number):
    print('\nReading Item by Id\n')

    # We can do an efficient point read lookup on partition key and id
    response = container.read_item(item=doc_id, partition_key=account_number)

    print('Item read by Id {0}'.format(doc_id))
    print('Partition Key: {0}'.format(response.get('partitionKey')))
    print('Subtotal: {0}'.format(response.get('subtotal')))


def read_items(container):
    print('\nReading all items in a container\n')

    # NOTE: Use MaxItemCount on Options to control how many items come back per
trip to the server
    #       Important to handle throttles whenever you are doing operations such
as this that might
    #       result in a 429 (throttled request)
    item_list = list(container.read_all_items(max_item_count=10))

    print('Found {0} items'.format(item_list.__len__()))

    for doc in item_list:
        print('Item Id: {0}'.format(doc.get('id')))


def query_items(container, account_number):
    print('\nQuerying for an  Item by Partition Key\n')

    # Including the partition key value of account_number in the WHERE filter
results in a more efficient query
    items = list(container.query_items(
        query="SELECT * FROM r WHERE r.partitionKey=@account_number",
        parameters=[
            { "name":"@account_number", "value": account_number }
        ]
    ))
```

```
    print('Item queried by Partition Key {0}'.format(items[0].get("id")))


def replace_item(container, doc_id, account_number):
    print('\nReplace an Item\n')

    read_item = container.read_item(item=doc_id, partition_key=account_number)
    read_item['subtotal'] = read_item['subtotal'] + 1
    response = container.replace_item(item=read_item, body=read_item)

    print('Replaced Item\'s Id is {0}, new subtotal={1}'.format(response['id'],
response['subtotal']))


def upsert_item(container, doc_id, account_number):
    print('\nUpserting an item\n')

    read_item = container.read_item(item=doc_id, partition_key=account_number)
    read_item['subtotal'] = read_item['subtotal'] + 1
    response = container.upsert_item(body=read_item)

    print('Upserted Item\'s Id is {0}, new subtotal={1}'.format(response['id'],
response['subtotal']))


def delete_item(container, doc_id, account_number):
    print('\nDeleting Item by Id\n')

    response = container.delete_item(item=doc_id, partition_key=account_number)

    print('Deleted item\'s Id is {0}'.format(doc_id))


def get_sales_order(item_id):
    order1 = {'id' : item_id,
            'partitionKey' : 'Account1',
            'purchase_order_number' : 'PO18009186470',
            'order_date' : datetime.date(2005,1,10).strftime('%c'),
            'subtotal' : 419.4589,
            'tax_amount' : 12.5838,
            'freight' : 472.3108,
            'total_due' : 985.018,
            'items' : [
                {'order_qty' : 1,
                    'product_id' : 100,
                    'unit_price' : 418.4589,
```

```
                    'line_price' : 418.4589
                }
                ],
            'ttl' : 60 * 60 * 24 * 30
            }

    return order1


def get_sales_order_v2(item_id):
    # notice new fields have been added to the sales order
    order2 = {'id' : item_id,
            'partitionKey' : 'Account2',
            'purchase_order_number' : 'PO15428132599',
            'order_date' : datetime.date(2005,7,11).strftime('%c'),
            'due_date' : datetime.date(2005,7,21).strftime('%c'),
            'shipped_date' : datetime.date(2005,7,15).strftime('%c'),
            'subtotal' : 6107.0820,
            'tax_amount' : 586.1203,
            'freight' : 183.1626,
            'discount_amt' : 1982.872,
            'total_due' : 4893.3929,
            'items' : [
                {'order_qty' : 3,
                    'product_code' : 'A-123',      # notice how in item details
we no longer reference a ProductId
                    'product_name' : 'Product 1',  # instead we have decided to
denormalise our schema and include
                    'currency_symbol' : '$',       # the Product details relevant
to the Order on to the Order directly
                    'currency_code' : 'USD',       # this is a typical refactor
that happens in the course of an application
                    'unit_price' : 17.1,           # that would have previously
required schema changes and data migrations etc.
                    'line_price' : 5.7
                }
                ],
            'ttl' : 60 * 60 * 24 * 30
            }

    return order2

def run_sample():
    client = cosmos_client.CosmosClient(HOST, {'masterKey': MASTER_KEY},
user_agent="CosmosDBPythonQuickstart", user_agent_overwrite=True)
```

```
    try:
        # setup database for this sample
        try:
            db = client.create_database(id=DATABASE_ID)
            print('Database with id \'{0}\' created'.format(DATABASE_ID))

        except exceptions.CosmosResourceExistsError:
            db = client.get_database_client(DATABASE_ID)
            print('Database with id \'{0}\' was found'.format(DATABASE_ID))

        # setup container for this sample
        try:
            container = db.create_container(id=CONTAINER_ID,
partition_key=PartitionKey(path='/partitionKey'))
            print('Container with id \'{0}\' created'.format(CONTAINER_ID))

        except exceptions.CosmosResourceExistsError:
            container = db.get_container_client(CONTAINER_ID)
            print('Container with id \'{0}\' was found'.format(CONTAINER_ID))

        scale_container(container)
        create_items(container)
        read_item(container, 'SalesOrder1', 'Account1')
        read_items(container)
        query_items(container, 'Account1')
        replace_item(container, 'SalesOrder1', 'Account1')
        upsert_item(container, 'SalesOrder1', 'Account1')
        delete_item(container, 'SalesOrder1', 'Account1')

        # cleanup database after sample
        try:
            client.delete_database(db)

        except exceptions.CosmosResourceNotFoundError:
            pass

    except exceptions.CosmosHttpResponseError as e:
        print('\nrun_sample has caught an error. {0}'.format(e.message))

    finally:
            print("\nrun_sample done")


if __name__ == '__main__':
    run_sample()
```