Microsoft

Developing Solutions for Microsoft Azure

THIRD EDITION

Exam Ref AZ-204

Santiago Fernández Muñoz

Microsoft

Exam Ref AZ-204 Developing Solutions for Microsoft Azure, Third Edition

Santiago Fernández Muñoz

- **UniqueKeyPolicy** You can configure which item's property is used as the unique key. Using unique keys, you ensure that you cannot insert two items with the same value for the same item. Keep in mind that the uniqueness is scoped to the logical partition. For example, if your item has the properties email, firstname, lastname, and company, and you define email as the unique key and company as the partition key, you cannot insert an item with the same email and company values. You can also create compound unique keys, such as email and firstname. Once you have created a unique key, you cannot change it. You can only define the unique key during the creation process of the container.
- **AnalyticalTimeToLive** Sets the time that an item will be kept in an analytical store container. The item is deleted from the container after the time specified in this property is reached. An analytical store is a special type of column store that is schematized to optimize for analytical query performance.

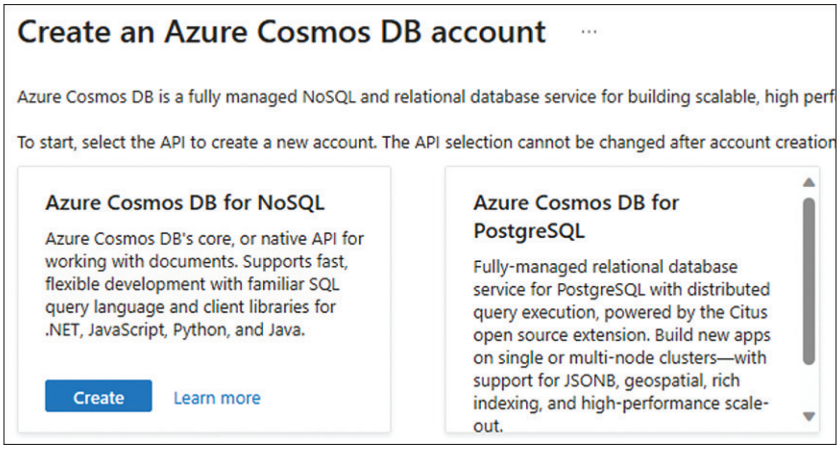
Apart from the properties that you reviewed in the previous list, there is a group of properties that are automatically generated and managed by the system. You can read these system-generated properties, but you cannot modify them. These properties are: `_rid`, `_etag`, `_ts`, and `_self`.

NOTE **CONTAINER PROPERTIES**

The properties available to the containers depend on the API you configured for your Azure Cosmos DB account. For a complete list of properties available for each API, please review the article at *https://docs.microsoft.com/en-us/azure/cosmos-db/databases-containers-items#azure-cosmos-containers*.

Before starting with the examples, you must create a Cosmos DB account to store your data. The following procedure shows how to create a Cosmos DB free account with the SQL API. You can use this same procedure for creating accounts with the other APIs reviewed in this skill:

1. Sign in to the Azure portal (*http://portal.azure.com*).
2. In the top left of the Azure portal, click the menu icon represented by three horizontal bars, and then select Create A Resource.
3. On the Create A Resource panel, under the Categories column, select Databases. On the Popular Azure Services column, click the Create link under Azure Cosmos DB.
4. On the Create An Azure Cosmos DB Account blade, click the Create button in the Azure Cosmos DB For NoSQL section, as shown in Figure 2-1.



Create an Azure Cosmos DB account ···

Azure Cosmos DB is a fully managed NoSQL and relational database service for building scalable, high perf

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.

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.

FIGURE 2-1 Selecting a Cosmos DB API

5. On the Create Azure Cosmos DB Account blade, in the Resource Group dropdown menu, click the Create New link below the dropdown menu. Type a name for the new Resource Group in the pop-up dialog box. Alternatively, you can select an existing Resource Group from the dropdown menu.
6. In the Instance Details section, type an Account Name.
7. On the Location dropdown menu, select the region most appropriate for you. If you are using App Services or virtual machines, select the region in which you deployed those services.
8. In the Capacity mode selection control, keep Provisioned throughput selected.
9. Ensure that the Apply Free Tier Discount switch is set to Apply.
10. Click the Next: Global Distribution button at the bottom of the blade.
11. Leave Geo-Redundancy, Multi-Region Write, and Availability Zones disabled.
12. Leave all other options in the other tabs with their default values.
13. In the bottom left of the Create An Azure Cosmos DB Account blade, click the Review + Create button.
14. In the bottom left of the Review + Create tab, click the Create button to start deploying your Cosmos DB account.

Once you have created an Azure Cosmos DB account, you can use the following procedure to create a new collection in your Cosmos DB account. This procedure might be slightly different depending on the API you choose for your Cosmos DB account. In this procedure, you use a Cosmos DB account configured with the NoSQL API:

1. Sign in to the Azure portal (*http://portal.azure.com*).
2. In the search box at the top of the Azure portal, type the name of your Cosmos DB account and then select your account name.
3. On your Cosmos DB account blade, select Data Explorer.

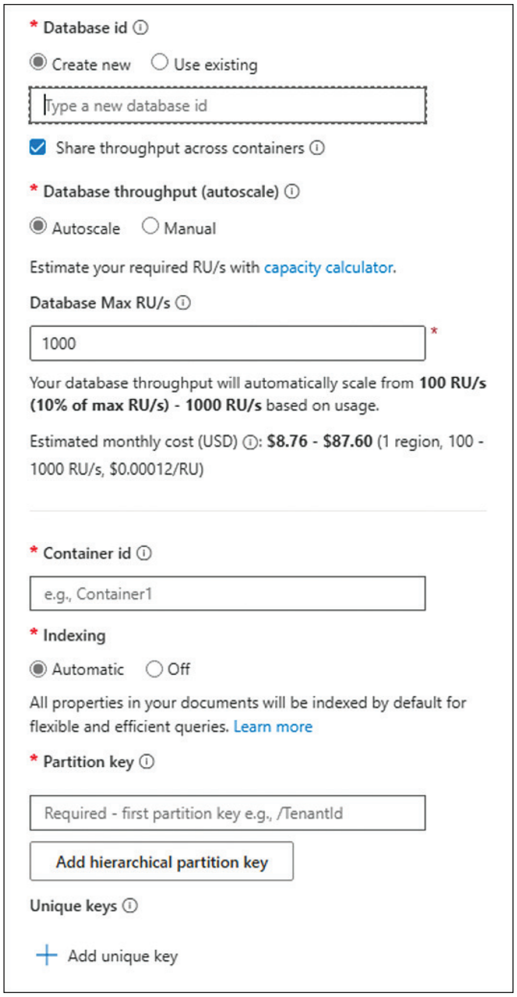
4. On the Data Explorer blade, click the New Container icon in the top left of the blade.
5. On the New Container panel, shown in Figure 2-2, provide a name for the new database. If you want to add a container to an existing database, you can select the database by clicking the Use Existing radio button.
6. Ensure that the Share Throughput Across Containers checkbox is selected. You are configuring this container as a shared throughput container using this option. If you want to create a dedicated throughput container, uncheck this option.
7. Leave the Database Throughput (Autoscale) value set to Autoscale. This is the value for the database throughput if the previous option is checked. Otherwise, this value represents the dedicated throughput reserved for the container.
8. Leave the Database Max RU/s value set to 1000. This is the maximum value of Request Units per second configured for your container. The capacity is scaled between the minimum 10 percent of the configured value and the maximum value. This option appears only if you select the Autoscale option for the Database Throughput setting.
9. In the Container ID text box, type a name for the container.
10. Keep the Indexing setting as Automatic.
11. Type a partition key in the Partition Key text box. The partition key must start with the slash character.
12. If you want to create a unique key for this container, click the Add Unique Key button.
13. Click the OK button at the bottom of the panel.

Estimated monthly cost (USD). This cost is an estimate and may vary based on the regions where your account is deployed and potential discounts applied to your account: $8.76 - $87.60 (1 region, 100 - 1000 RU/s, $0.00012/RU)" The Container ID setting shows a blank text box into which you can type a new Container ID. The Indexing option offers the options Automatic and Off. The Automatic option is selected. The last setting, Partition Key, does not show any options. An Add Unique Key button appears at the bottom of the dialog box.

NEED MORE REVIEW? **TIME TO LIVE, INDEXES, AND CHANGES FEED**

You can review the details of how to configure the Time To Live, Index Policies, and Changes Feed by reading the following articles:

- **Configure Time to Live in Azure Cosmos DB** ***https://docs.microsoft.com/en-us/azure/cosmos-db/how-to-time-to-live***
- **Unique Key Constraints in Azure Cosmos DB** ***https://docs.microsoft.com/en-us/azure/cosmos-db/unique-keys***
- **Change Feed Design Patterns in Azure Cosmos DB** ***https://docs.microsoft.com/en-us/azure/cosmos-db/change-feed-design-patterns***


* Database id ⓘ

(● Create new) ○ Use existing

Type a new database id

✓ Share throughput across containers ⓘ

* Database throughput (autoscale) ⓘ

(● Autoscale) ○ Manual

Estimate your required RU/s with capacity calculator.

Database Max RU/s ⓘ

1000 *

Your database throughput will automatically scale from **100 RU/s (10% of max RU/s) - 1000 RU/s** based on usage.

Estimated monthly cost (USD) ⓘ: **$8.76 - $87.60** (1 region, 100 - 1000 RU/s, $0.00012/RU)

* Container id ⓘ

e.g., Container1

* Indexing

(● Automatic) ○ Off

All properties in your documents will be indexed by default for flexible and efficient queries. Learn more

* Partition key ⓘ

Required - first partition key e.g., /TenantId

Unique keys ⓘ

FIGURE 2-2 Creating a new collection

EXAM TIP

You must plan carefully how to create a new container in Azure Cosmos DB. You can configure some of the properties only during creation. Once you have created the container, if you need to modify those properties, you must create a new container with the needed values and migrate the data to the new container.

Once you have configured your container, you can create items on it. As mentioned in this section, you can use different languages, such as .NET, Node.js, Java, Python, or Go.

The following example shows how to create a console application using .NET Core. The first example uses Cosmos DB SQL API for creating, updating, and deleting some elements in the Cosmos DB account:

1. Open Visual Studio Code and create a directory for storing the example project.
2. Open the Terminal, switch to the project's directory, and type the following command:

   ```
   dotnet new console
   ```

3. Install the NuGet package using the SQL API to interact with your Cosmos DB account. Type the following command in the Terminal:

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

4. Change the content of the Program.cs file using the content provided in Listing 2-2. You need to change the namespace according to your project's name.
5. Sign in to the Azure portal (*http://portal.azure.com*).
6. In the search box at the top of the Azure portal, type the name of your Cosmos DB account and then click the name of the account.
7. On your Cosmos DB Account blade, in the Settings section, select Keys.
8. On the Keys panel, copy the URI and Primary Keys values from the Read-Write Keys tab. You need to provide these values to the EndpointUri and Key Constants in the code shown in Listing 2-2. (The most important parts of the code are shown with bold format.)

LISTING 2-2 Cosmos DB NoSQL API example

```
//C# .NET 6.0 LTS. Program.cs
using System.Collections.Immutable;
using System.Xml.Linq;
using System.Diagnostics;
using System.Runtime.CompilerServices;
using System;

using System.Linq;
using Microsoft.Azure.Cosmos;
using System.Threading.Tasks;
using ch2_1_1_NoSQL.Model;
using System.Net;

namespace ch2_1_1_NoSQL
{
    class Program
    {
        private const string EndpointUri = "<PUT YOUR ENDPOINT URL HERE>";
        private const string Key = "<PUT YOUR COSMOS DB KEY HERE>";
        private CosmosClient client;
        private Database database;
        private Container container;

        static void Main(string[] args)
        {
```

```
            try
            {
                Program demo = new Program();
                demo.StartDemo().Wait();
            }
            catch (CosmosException ce)
            {
                Exception baseException = ce.GetBaseException();
                System.Console.WriteLine($"{ce.StatusCode} error ocurred:
                {ce.Message}, Message: {baseException.Message}");
            }
            catch (Exception ex)
            {
                Exception baseException = ex.GetBaseException();
                System.Console.WriteLine($"Error ocurred: {ex.Message},
Message: {baseException.Message}");
            }

        }

        private async Task StartDemo()
        {
            Console.WriteLine("Starting Cosmos DB NoSQL API Demo!");

            //Create a new demo database

            string databaseName = "demoDB_" + Guid.NewGuid().ToString().
Substring(0, 5);

            this.SendMessageToConsoleAndWait($"Creating database {databaseName}...");

            this.client = new CosmosClient(EndpointUri, Key);
            this.database = await this.client.CreateDatabaseIfNotExistsAsync
(databaseName);

            //Create a new demo collection inside the demo database.
            //This creates a collection with a reserved throughput. You can customize
            //the options using a ContainerProperties object
            //This operation has pricing implications.
            string containerName = "collection_" + Guid.NewGuid().ToString().
Substring(0, 5);


            this.SendMessageToConsoleAndWait($"Creating collection demo
{containerName}...");

            this.container = await this.database.CreateContainerIfNotExistsAsync
(containerName, "/LastName");

            //Create some documents in the collection
            Person person1 = new Person
            {
                Id = "Person.1",
                FirstName = "Santiago",
                LastName = "Fernandez",
```