100 XP

Introduction

3 minutes

Azure Cosmos DB is a globally distributed database system that allows you to read and write data from the local replicas of your database and it transparently replicates the data to all the regions associated with your Cosmos account.

After completing this module, you'll be able to:

- Identify the key benefits provided by Azure Cosmos DB
- Describe the elements in an Azure Cosmos DB account and how they are organized
- Explain the different consistency levels and choose the correct one for your project
- Explore the APIs supported in Azure Cosmos DB and choose the appropriate API for your solution
- Describe how request units impact costs
- Create Azure Cosmos DB resources by using the Azure portal.

Next unit: Identify key benefits of Azure Cosmos DB

Continue

100 XP

Identify key benefits of Azure Cosmos DB

3 minutes

Azure Cosmos DB is designed to provide low latency, elastic scalability of throughput, well-defined semantics for data consistency, and high availability.

You can configure your databases to be globally distributed and available in any of the Azure regions. To lower the latency, place the data close to where your users are. Choosing the required regions depends on the global reach of your application and where your users are located.

With Azure Cosmos DB, you can add or remove the regions associated with your account at any time. Your application doesn't need to be paused or redeployed to add or remove a region.

Key benefits of global distribution

With its novel multi-master replication protocol, every region supports both writes and reads. The multi-master capability also enables:

- Unlimited elastic write and read scalability.
- 99.999% read and write availability all around the world.
- Guaranteed reads and writes served in less than 10 milliseconds at the 99th percentile.

Your application can perform near real-time reads and writes against all the regions you chose for your database. Azure Cosmos DB internally handles the data replication between regions with consistency level guarantees of the level you've selected.

Running a database in multiple regions worldwide increases the availability of a database. If one region is unavailable, other regions automatically handle application requests. Azure Cosmos DB offers 99.999% read and write availability for multi-region databases.

Next unit: Explore the resource hierarchy

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore the resource hierarchy

3 minutes

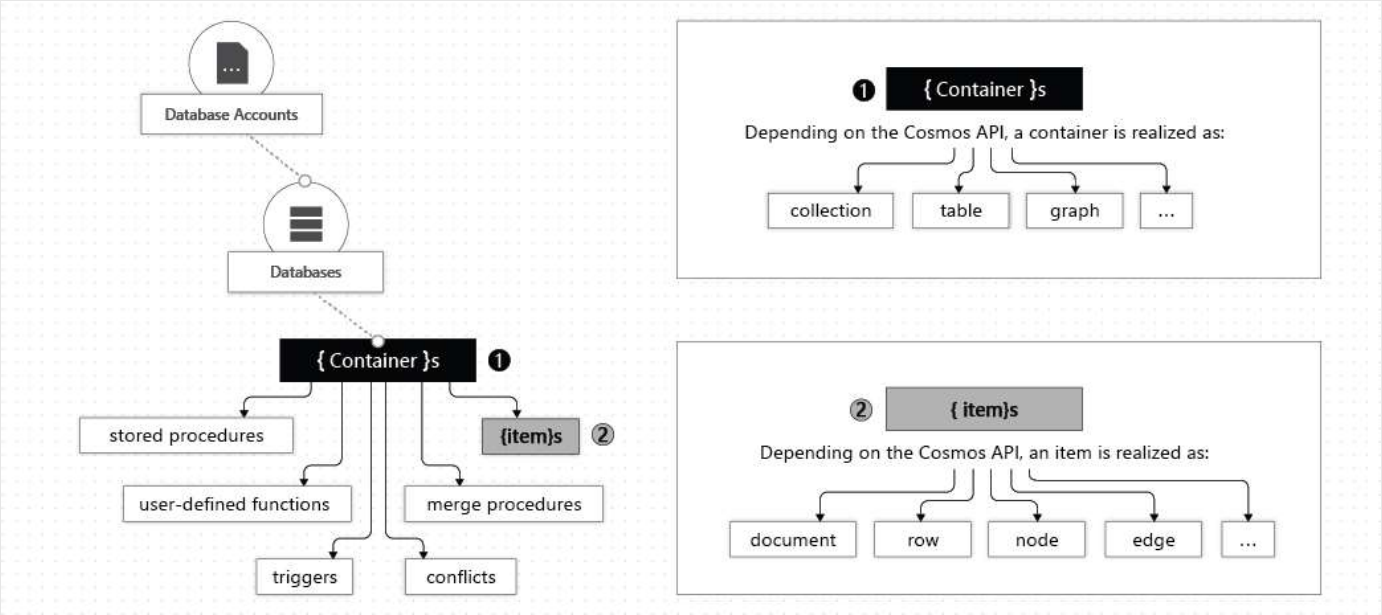
The Azure Cosmos account is the fundamental unit of global distribution and high availability. Your Azure Cosmos account contains a unique DNS name and you can manage an account by using the Azure portal or the Azure CLI, or by using different language-specific SDKs. For globally distributing your data and throughput across multiple Azure regions, you can add and remove Azure regions to your account at any time.

Elements in an Azure Cosmos account

An Azure Cosmos container is the fundamental unit of scalability. You can virtually have an unlimited provisioned throughput (RU/s) and storage on a container. Azure Cosmos DB transparently partitions your container using the logical partition key that you specify in order to elastically scale your provisioned throughput and storage.

Currently, you can create a maximum of 50 Azure Cosmos accounts under an Azure subscription (this is a soft limit that can be increased via support request). After you create an account under your Azure subscription, you can manage the data in your account by creating databases, containers, and items.

The following image shows the hierarchy of different entities in an Azure Cosmos DB account:



Azure Cosmos databases

You can create one or multiple Azure Cosmos databases under your account. A database is analogous to a namespace. A database is the unit of management for a set of Azure Cosmos containers. The following table shows how an Azure Cosmos database is mapped to various API-specific entities:

Azure Cosmos entity	SQL API	Cassandra API	Azure Cosmos DB API for MongoDB	Gremlin API	Table API
Azure Cosmos database	Database	Keyspace	Database	Database	NA

> **Note**
>
> With Table API accounts, when you create your first table, a default database is automatically created in your Azure Cosmos account.

Azure Cosmos containers

An Azure Cosmos container is the unit of scalability both for provisioned throughput and storage. A container is horizontally partitioned and then replicated across multiple regions. The items that you add to the container are automatically grouped into logical partitions, which are distributed

across physical partitions, based on the partition key. The throughput on a container is evenly distributed across the physical partitions.

When you create a container, you configure throughput in one of the following modes:

- **Dedicated provisioned throughput mode**: The throughput provisioned on a container is exclusively reserved for that container and it is backed by the SLAs.
- **Shared provisioned throughput mode**: These containers share the provisioned throughput with the other containers in the same database (excluding containers that have been configured with dedicated provisioned throughput). In other words, the provisioned throughput on the database is shared among all the "shared throughput" containers.

A container is a schema-agnostic container of items. Items in a container can have arbitrary schemas. For example, an item that represents a person and an item that represents an automobile can be placed in the same container. By default, all items that you add to a container are automatically indexed without requiring explicit index or schema management.

Azure Cosmos items

Depending on which API you use, an Azure Cosmos item can represent either a document in a collection, a row in a table, or a node or edge in a graph. The following table shows the mapping of API-specific entities to an Azure Cosmos item:

Cosmos entity	SQL API	Cassandra API	Azure Cosmos DB API for MongoDB	Gremlin API	Table API
Azure Cosmos item	Item	Row	Document	Node or edge	Item

Next unit: Explore consistency levels

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore consistency levels

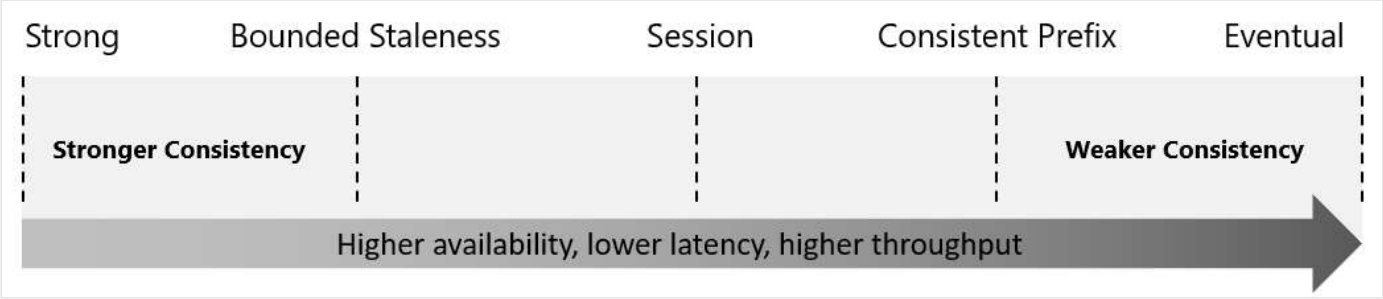
3 minutes

Azure Cosmos DB approaches data consistency as a spectrum of choices instead of two extremes. Strong consistency and eventual consistency are at the ends of the spectrum, but there are many consistency choices along the spectrum. Developers can use these options to make precise choices and granular tradeoffs with respect to high availability and performance.

With Azure Cosmos DB, developers can choose from five well-defined consistency models on the consistency spectrum. From strongest to more relaxed, the models include:

- *strong*
- *bounded staleness*
- *session*
- *consistent prefix*
- *eventual*

Each level provides availability and performance tradeoffs. The following image shows the different consistency levels as a spectrum.



The consistency levels are region-agnostic and are guaranteed for all operations regardless of the region from which the reads and writes are served, the number of regions associated with your Azure Cosmos account, or whether your account is configured with a single or multiple write regions.

Read consistency applies to a single read operation scoped within a partition-key range or a logical partition. The read operation can be issued by a remote client or a stored procedure.

Next unit: Choose the right consistency level

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Choose the right consistency level

3 minutes

Azure Cosmos DB allows developers to choose among the five consistency models: strong, bounded staleness, session, consistent prefix and eventual. Each of these consistency models can be used for specific real-world scenarios. Each provides precise availability and performance tradeoffs and are backed by comprehensive SLAs. The following simple considerations will help you make the right choice in many common scenarios.

SQL API and Table API

Consider the following points if your application is built using SQL API or Table API:

- For many real-world scenarios, session consistency is optimal and it's the recommended option.
- If your application requires strong consistency, it is recommended that you use bounded staleness consistency level.
- If you need stricter consistency guarantees than the ones provided by session consistency and single-digit-millisecond latency for writes, it is recommended that you use bounded staleness consistency level.
- If your application requires eventual consistency, it is recommended that you use consistent prefix consistency level.
- If you need less strict consistency guarantees than the ones provided by session consistency, it is recommended that you use consistent prefix consistency level.
- If you need the highest throughput and the lowest latency, then use eventual consistency level.
- If you need even higher data durability without sacrificing performance, you can create a custom consistency level at the application layer.

Cassandra, MongoDB, and Gremlin APIs

Azure Cosmos DB provides native support for wire protocol-compatible APIs for popular databases. These include MongoDB, Apache Cassandra, and Gremlin. When using Gremlin API the default consistency level configured on the Azure Cosmos account is used. For details on consistency level mapping between Cassandra API or the API for MongoDB and Azure Cosmos DB's consistency levels see, Cassandra API consistency mapping and API for MongoDB consistency mapping.

Consistency guarantees in practice

In practice, you may often get stronger consistency guarantees. Consistency guarantees for a read operation correspond to the freshness and ordering of the database state that you request. Read-consistency is tied to the ordering and propagation of the write/update operations.

- When the consistency level is set to **bounded staleness**, Cosmos DB guarantees that the clients always read the value of a previous write, with a lag bounded by the staleness window.
- When the consistency level is set to **strong**, the staleness window is equivalent to zero, and the clients are guaranteed to read the latest committed value of the write operation.
- For the remaining three consistency levels, the staleness window is largely dependent on your workload. For example, if there are no write operations on the database, a read operation with **eventual**, **session**, or **consistent prefix** consistency levels is likely to yield the same results as a read operation with strong consistency level.

If your Azure Cosmos account is configured with a consistency level other than the strong consistency, you can find out the probability that your clients may get strong and consistent reads for your workloads by looking at the *Probabilistically Bounded Staleness* (PBS) metric.

Probabilistic bounded staleness shows how eventual your eventual consistency is. This metric provides an insight into how often you can get a stronger consistency than the consistency level that you have currently configured on your Azure Cosmos account. In other words, you can see the probability (measured in milliseconds) of getting strongly consistent reads for a combination of write and read regions.

Next unit: Explore supported APIs

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore supported APIs

3 minutes

Azure Cosmos DB offers multiple database APIs, which include the Core (SQL) API, API for MongoDB, Cassandra API, Gremlin API, and Table API. By using these APIs, you can model real world data using documents, key-value, graph, and column-family data models. These APIs allow your applications to treat Azure Cosmos DB as if it were various other databases technologies, without the overhead of management, and scaling approaches.

Core(SQL) API

This API stores data in document format. It offers the best end-to-end experience as we have full control over the interface, service, and the SDK client libraries. Any new feature that is rolled out to Azure Cosmos DB is first available on SQL API accounts. Azure Cosmos DB SQL API accounts provide support for querying items using the Structured Query Language (SQL) syntax, one of the most familiar and popular query languages.

If you are migrating from other databases such as Oracle, DynamoDB, HBase etc. SQL API is the recommended option. SQL API supports analytics and offers performance isolation between operational and analytical workloads.

API for MongoDB

This API stores data in a document structure, via BSON format. It is compatible with MongoDB wire protocol; however, it does not use any native MongoDB related code. This API is a great choice if you want to use the broader MongoDB ecosystem and skills, without compromising on using Azure Cosmos DB's features such as scaling, high availability, geo-replication, multiple write locations, automatic and transparent shard management, transparent replication between operational and analytical stores, and more. You can use your existing MongoDB apps with API for MongoDB by just changing the connection string.

API for MongoDB is compatible with the 4.0, 3.6, and 3.2 MongoDB server versions. Server version 4.0 is recommended as it offers the best performance and full feature support

Cassandra API

This API stores data in column-oriented schema. Cassandra API is wire protocol compatible with the Apache Cassandra. You should consider Cassandra API if you want to benefit the elasticity and fully managed nature of Azure Cosmos DB and still use most of the native Apache Cassandra features, tools, and ecosystem. This means on Cassandra API you don't need to manage the OS, Java VM, garbage collector, read/write performance, nodes, clusters, etc.

You can use Apache Cassandra client drivers to connect to the Cassandra API. The Cassandra API enables you to interact with data using the Cassandra Query Language (CQL), and tools like CQL shell, Cassandra client drivers that you're already familiar with. Cassandra API currently only supports OLTP scenarios. Using Cassandra API, you can also use the unique features of Azure Cosmos DB such as change feed.

Table API

This API stores data in key/value format. If you are currently using Azure Table storage, you may see some limitations in latency, scaling, throughput, global distribution, index management, low query performance. Table API overcomes these limitations and it's recommended to migrate your app if you want to use the benefits of Azure Cosmos DB. Table API only supports OLTP scenarios.

Applications written for Azure Table storage can migrate to the Table API with little code changes and take advantage of premium capabilities.

Gremlin API

This API allows users to make graph queries and stores data as edges and vertices. Use this API for scenarios involving dynamic data, data with complex relations, data that is too complex to be modeled with relational databases, and if you want to use the existing Gremlin ecosystem and skills. Gremlin API currently only supports OLTP scenarios.

Next unit: Discover request units

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Discover request units

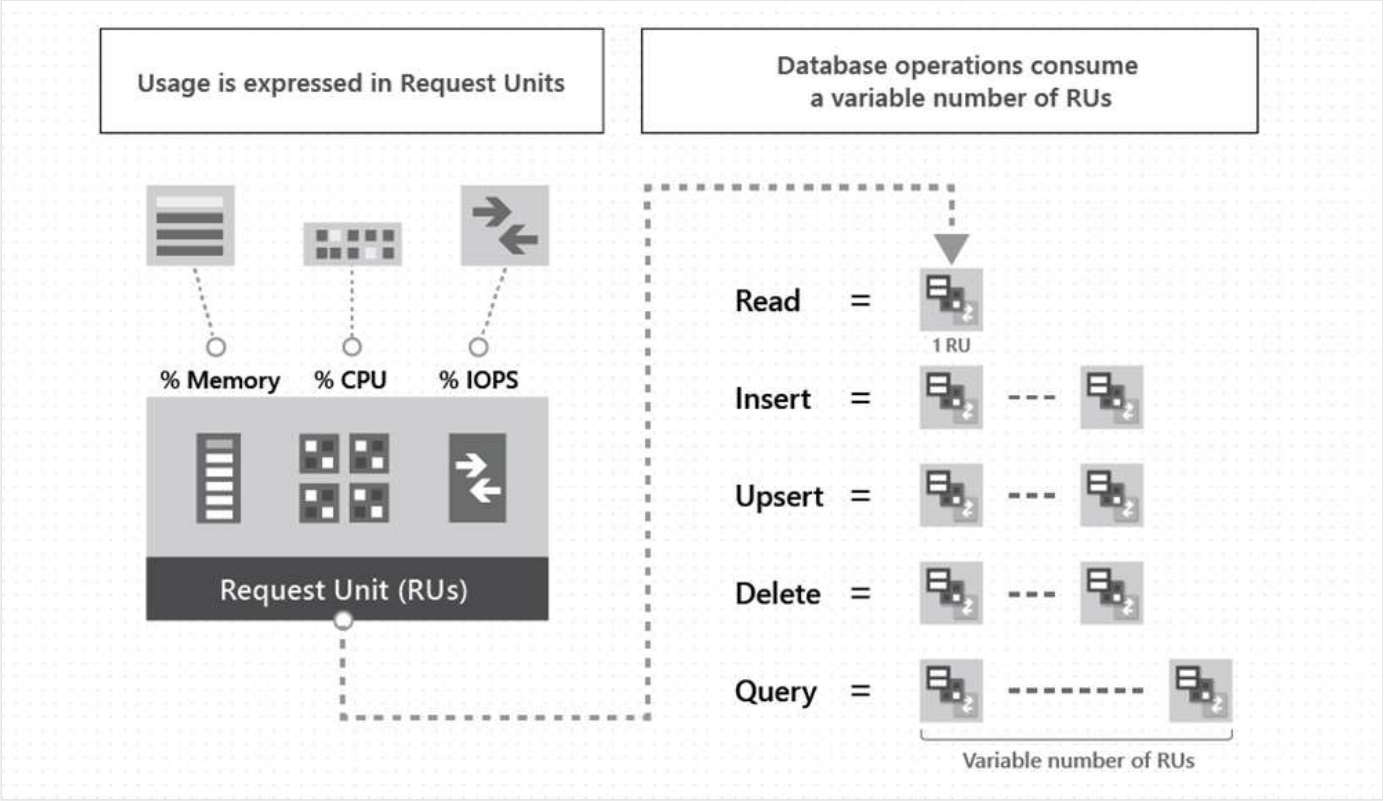
3 minutes

With Azure Cosmos DB, you pay for the throughput you provision and the storage you consume on an hourly basis. Throughput must be provisioned to ensure that sufficient system resources are available for your Azure Cosmos database at all times.

The cost of all database operations is normalized by Azure Cosmos DB and is expressed by *request units* (or RUs, for short). A request unit represents the system resources such as CPU, IOPS, and memory that are required to perform the database operations supported by Azure Cosmos DB.

The cost to do a point read, which is fetching a single item by its ID and partition key value, for a 1KB item is 1RU. All other database operations are similarly assigned a cost using RUs. No matter which API you use to interact with your Azure Cosmos container, costs are always measured by RUs. Whether the database operation is a write, point read, or query, costs are always measured in RUs.

The following image shows the high-level idea of RUs:



The type of Azure Cosmos account you're using determines the way consumed RUs get charged. There are three modes in which you can create an account:

- **Provisioned throughput mode**: In this mode, you provision the number of RUs for your application on a per-second basis in increments of 100 RUs per second. To scale the provisioned throughput for your application, you can increase or decrease the number of RUs at any time in increments or decrements of 100 RUs. You can make your changes either programmatically or by using the Azure portal. You can provision throughput at container and database granularity level.

- **Serverless mode**: In this mode, you don't have to provision any throughput when creating resources in your Azure Cosmos account. At the end of your billing period, you get billed for the amount of request units that has been consumed by your database operations.

- **Autoscale mode**: In this mode, you can automatically and instantly scale the throughput (RU/s) of your database or container based on it's usage. This mode is well suited for mission-critical workloads that have variable or unpredictable traffic patterns, and require SLAs on high performance and scale.

Next unit: Exercise: Create Azure Cosmos DB resources by using the Azure portal

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

Exercise: Create Azure Cosmos DB resources by using the Azure portal

10 minutes

In this exercise you'll learn how to perform the following actions in the Azure portal:

- Create an Azure Cosmos DB account
- Add a database and a container
- Add data to your database
- Clean up resources

Prerequisites

- An Azure account with an active subscription. If you don't already have one, you can sign up for a free trial at https://azure.com/free .

Create an Azure Cosmos DB account

1. Log in to the Azure portal .
2. From the Azure portal navigation pane, select + **Create a resource**.
3. Search for **Azure Cosmos DB**, then select **Create/Azure Cosmos DB** to get started.
4. On the Select API option page, select **Create** in the **Core (SQL) - Recommended** box.
5. In the **Create Azure Cosmos DB Account - Core (SQL)** page, enter the basic settings for the new Azure Cosmos account.
    - **Subscription**: Select the subscription you want to use.
    - **Resource Group**: Select **Create new**, then enter *az204-cosmos-rg*.
    - **Account Name**: Enter a *unique* name to identify your Azure Cosmos account. The name can only contain lowercase letters, numbers, and the hyphen (-) character. It must be between 3-31 characters in length.

- **Location**: Use the location that is closest to your users to give them the fastest access to the data.
- **Capacity mode**: Select **Serverless**.

6. Select **Review + create**.

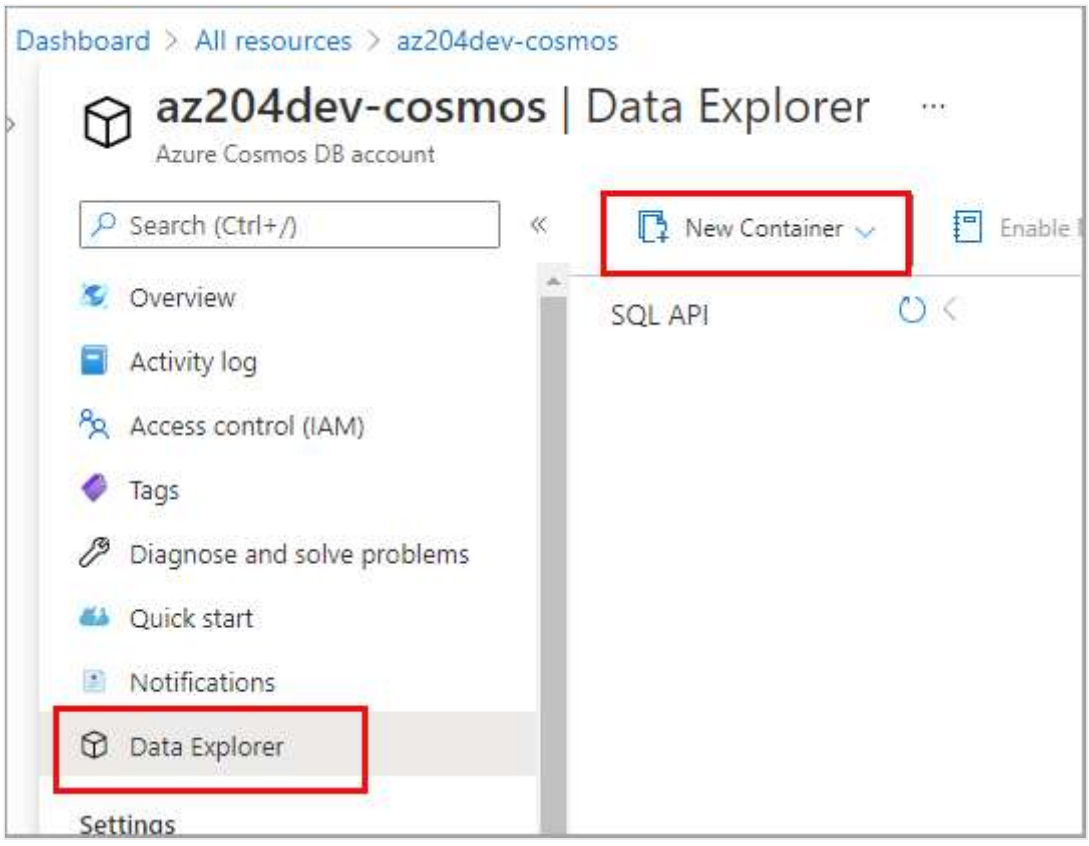
7. Review the account settings, and then select **Create**. It takes a few minutes to create the account. Wait for the portal page to display **Your deployment is complete**.

8. Select **Go to resource** to go to the Azure Cosmos DB account page.

Add a database and a container

You can use the Data Explorer in the Azure portal to create a database and container.

1. Select **Data Explorer** from the left navigation on your Azure Cosmos DB account page, and then select **New Container**.



2. In the **Add container** pane, enter the settings for the new container.

- **Database ID**: Select **Create new**, and enter *ToDoList*.
- **Container ID**: Enter *Items*

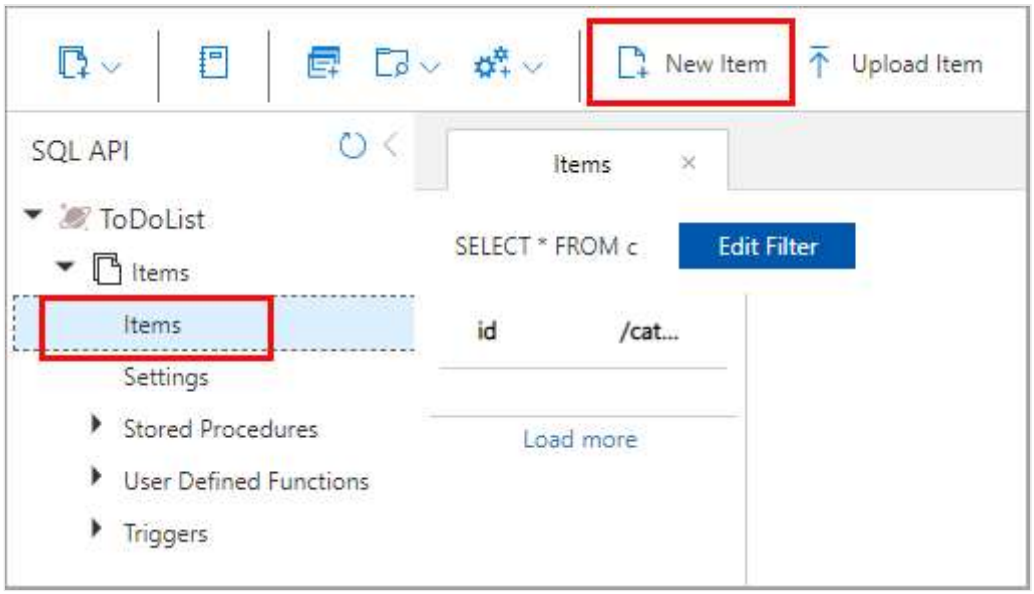
- **Partition key**: Enter */category*. The samples in this demo use */category* as the partition key.

3. Select **OK**. The Data Explorer displays the new database and the container that you created.

Add data to your database

Add data to your new database using Data Explorer.

1. In **Data Explorer**, expand the **ToDoList** database, and expand the **Items** container. Next, select **Items**, and then select **New Item**.



2. Add the following structure to the item on the right side of the **Items** pane:

JSON Copy

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

3. Select **Save**.

4. Select **New Item** again, and create and save another item with a unique `id`, and any other properties and values you want. Your items can have any structure, because Azure Cosmos DB doesn't impose any schema on your data.

Clean up resources

1. Select **Overview** from the left navigation on your Azure Cosmos DB account page.
2. Select the **az204-cosmos-rg** resource group link in the Essentials group.
3. Select **Delete** resource group and follow the directions to delete the resource group and all of the resources it contains.

200 XP

Knowledge check

5 minutes

Check your knowledge

1. When setting up Azure Cosmos DB there are three account type options. Which of the account type options below is used to specify the number of RUs for an application on a per-second basis?

- (●) Provisioned throughput
  - ✔ That's correct. In this mode, you provision the number of RUs for your application on a per-second basis in increments of 100 RUs per second.
- ( ) Serverless
- ( ) Autoscale

2. Which of the following consistency levels below offers the greatest throughput?

- ( ) Strong
- (●) Session
  - ✘ That's incorrect. The eventual consistency level offers the greatest throughput at the cost of weaker consistency.
- ( ) Eventual
  - ✔ That's correct. The eventual consistency level offers the greatest throughput at the cost of weaker consistency.

Next unit: Summary

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Summary

3 minutes

In this module you learned how to:

- Identify the key benefits provided by Azure Cosmos DB
- Describe the elements in an Azure Cosmos DB account and how they are organized
- Explain the different consistency levels and choose the correct one for your project
- Explore the APIs supported in Azure Cosmos DB and choose the appropriate API for your solution
- Describe how request units impact costs
- Create Azure Cosmos DB resources by using the Azure portal.

Module complete:

Unlock achievement

How are we doing? ☆ ☆ ☆ ☆ ☆