



# Azure Cosmos DB

Azure Cosmos DB is a **fully managed, globally distributed NoSQL database** service designed for **high availability, low latency, and scalability**. It supports multiple data models, including **document, key-value, graph, and column-family**, making it suitable for various applications.

## Use Cases of Azure Cosmos DB

### 1. IoT and Real-Time Analytics

- Stores and processes massive volumes of real-time sensor and device data.

### 2. E-commerce and Retail

- Manages product catalogs, order processing, and personalized recommendations.

### 3. Gaming Applications

- Supports leaderboards, game state management, and user profiles across regions.

### 4. Financial Services

- Handles fraud detection, transaction processing, and personalized banking.

### 5. Healthcare and Medical Records

- Stores patient records, medical history, and real-time health monitoring data.

### 6. Social Media and Messaging Apps

- Supports chat applications, user interactions, and content recommendations.

## Benefits of Azure Cosmos DB

### 1. Global Distribution

- Automatically replicates data across multiple regions with low latency.

### 2. Multi-Model Database

- Supports document (JSON), graph, key-value, column-family, and table storage.

### 3. Guaranteed High Performance

- Provides **millisecond response times** and **99.999% availability** SLA.

### 4. Elastic Scalability

- Automatically scales storage and throughput as demand increases.

### 5. Multi-API Support

- Works with SQL, MongoDB, Cassandra, Gremlin (graph), and Table APIs.

## 6. Serverless and Provisioned Throughput

- Offers flexible pricing models, optimizing cost for workloads.

## 7. Security and Compliance

- Provides **encryption at rest and in transit**, role-based access, and compliance with GDPR, HIPAA, etc.

## Conclusion

Azure Cosmos DB is ideal for **high-scale, globally distributed applications** requiring **low-latency and high availability**. Its flexibility, scalability, and multi-model support make it perfect for **IoT, financial services, gaming, and e-commerce** applications.

In this lab, we set up Azure Cosmos DB for NoSQL to store and manage JSON-based data. We begin by creating a Cosmos DB account in the Azure Portal, selecting NoSQL API, and configuring basic settings. After deployment, we use Data Explorer to create a database and a container with a partition key. We then manually add order-related JSON objects to the container, representing structured data. The end goal is to demonstrate how to store, retrieve, and manage NoSQL data efficiently in Azure Cosmos DB, providing a scalable and flexible database solution for various applications.

## 😊 To begin with the Lab

1. In this lab, we are going to work with Azure Cosmos DB. So, in the Azure Portal from the market place search for Azure Cosmos DB.



2. After clicking on Create button you will have two options to choose from. You have to choose Cosmos DB for NoSQL.

## 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.

[Recommended APIs](#)   [Others](#)

#### 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 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](#)

- Now in the workload type choose learning, and select your subscription. Give a unique name to your Cosmos DB account. Then just leave every setting as it is and create your Cosmos DB.

#### Project Details

Choose a workload type that best aligns with your goals. This helps us provide an optimized starting point for your Azure Cosmos DB account setup. or stick to the defaults provided.

Workload Type \* ⓘ

Learning

Best for beginners. Low cost, easy setup, and quick exploration to learn Azure Cosmos DB.

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

Subscription \*

MSDN Platforms Subscription

Resource Group \*

NewRG

[Create new](#)

#### Instance Details

Account Name \*

cosmosdb1231

Configure availability zone settings for your account. You cannot change these settings once the account is created.

Availability Zones ⓘ

Enable  Disable

- Once the deployment is complete, move to the resource.
- Once you are inside the Cosmos DB open the data explorer. We can create now something known as a database within the account, and we can create something known as a container within the database.

Welcome to Azure Cosmos DB

Globally distributed, multi-model database service for any scale

Launch quick start

New Container

Connect

6. Now you need to click on the new database to create a new one.

+ New Container

+ New Database

7. Give a name to your database ID and then in the provision throughput choose manual because we are not going to use much of the request units. Click on OK.

\* Database id ⓘ

demodb

Provision throughput ⓘ

\* Database throughput (400 - unlimited RU/s) ⓘ

Autoscale  Manual

Estimate your required RU/s with [capacity calculator](#).

Database Required RU/s ⓘ

400

\*

Estimated cost (USD) ⓘ: **\$0.032 hourly / \$0.77 daily / \$23.36**

**monthly** (1 region, 400RU/s, \$0.00008/RU)

---

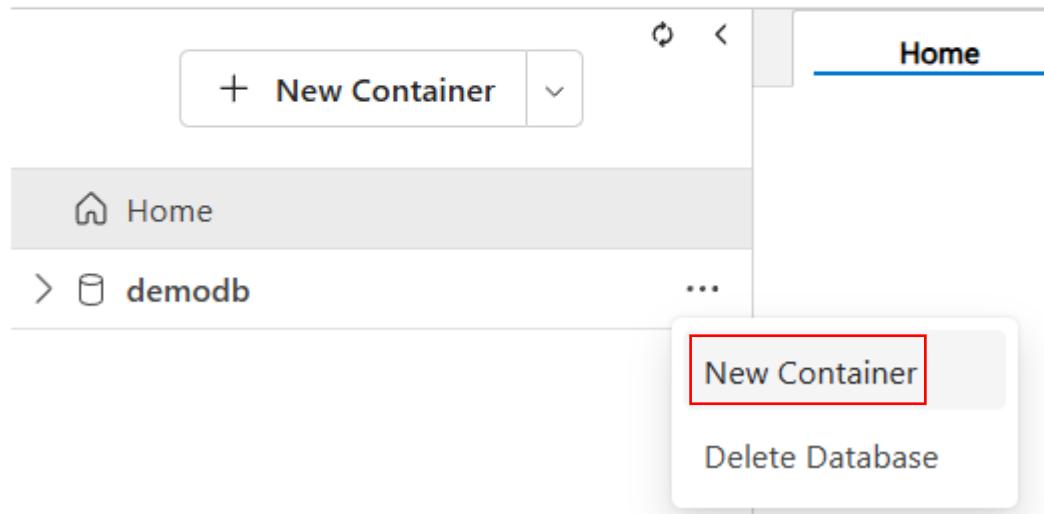
OK

8. Within the database now we can create a new container.

# cosmosdb1231 | Data Explorer ☆ ...

Azure Cosmos DB account

»  Enable Azure Synapse Link



The screenshot shows the Azure Cosmos DB Data Explorer interface. At the top, there's a navigation bar with a cube icon, the account name "cosmosdb1231", and a "Data Explorer" tab. Below the navigation bar, there's a breadcrumb trail: "Home > demodb". On the left, there's a "New Container" button. A context menu is open over the "demodb" entry, with two options: "New Container" (which is highlighted with a red box) and "Delete Database".

9. Here first we will give a container ID then the partition key. After that click on Ok.

## New Container

X

**\* Database id** ⓘ

Create new  Use existing

demodb



**\* Container id** ⓘ

orders

**\* Indexing**

Automatic  Off

All properties in your documents will be indexed by default for flexible and efficient queries. [Learn more](#)

**\* Partition key** ⓘ

/category

[Add hierarchical partition key](#)

Provision dedicated throughput for this container ⓘ

10. Within the DB you have the container now. Click on Items to add a new item.

 Home

✓  demodb

 Scale

✓  orders

 Items

 Settings

>  Stored Procedures

>  User Defined Functions

>  Triggers

11. We are going to be storing information about orders. When you choose the NoSQL API, you add data in the form of JSON-based objects. Here, just as an example, we have four objects in place. Each object has an order ID, a category, and a quantity.

```
1  {
2      "orderId":"01",
3      "category":"Laptop",
4      "quantity":100
5  }
6
7
8  {
9      "orderId":"02",
10     "category":"Mobiles",
11     "quantity":200
12  }
13
14
15  {
16      "orderId":"03",
17      "category":"Desktop",
18      "quantity":75
19  }
20
21
22  {
23      "orderId":"04",
24      "category":"Laptop",
25      "quantity":25
26  }
```

```
{
    "orderId":"O1",
    "category":"Laptop",
    "quantity":100
}
```

```
{
    "orderId":"O2",
    "category":"Mobiles",
    "quantity":200
}
```

```
{  
    "orderId": "O3",  
    "category": "Desktop",  
    "quantity": 75  
}
```

```
{  
    "orderId": "O4",  
    "category": "Laptop",  
    "quantity": 25  
}
```

12. Now click on new to item to add an item in the database.

The screenshot shows the Azure Cosmos DB Data Explorer interface. At the top, there's a navigation bar with icons for Home, Databases, Containers, and Functions, followed by the account name 'cosmosdb1231 | Data Explorer'. Below the navigation bar is a toolbar with 'New Item' and 'Upload Item' buttons. The main area shows a hierarchical tree on the left with 'demodb' expanded, showing 'orders' and 'orders.Items'. The 'orders.Items' node is selected and highlighted in grey. To the right of the tree is a query editor window with the query 'SELECT \* FROM c'. Below the query editor is a table preview showing columns: id, ... (ellipsis), and /category. The 'id' column has a checkbox next to it. The table currently displays one row with the value 'orders.Items' in the id column.

13. You have to manually add every item and click on save button to save your item in the database.

The screenshot shows the Azure Cosmos DB Data Explorer interface. On the left, there's a navigation sidebar with options like Home, demodb (with Scale), orders (with Items selected), Settings, Stored Procedures, User Defined Functions, and Triggers. The main area is titled "orders.items" and contains a table with the following data:

	id	/category	...
1	1783a224-c200-4a1e-adb8-0...	Laptop	
2			
3			
4			
5			

A code editor at the top shows the query: "SELECT \* FROM c". To the right of the table, a JSON representation of the data is shown:

```
[{"orderId": "02", "category": "Mobiles", "quantity": 200}]
```

14. Below you can see that we have items added to our NoSQL Database.