😊 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.

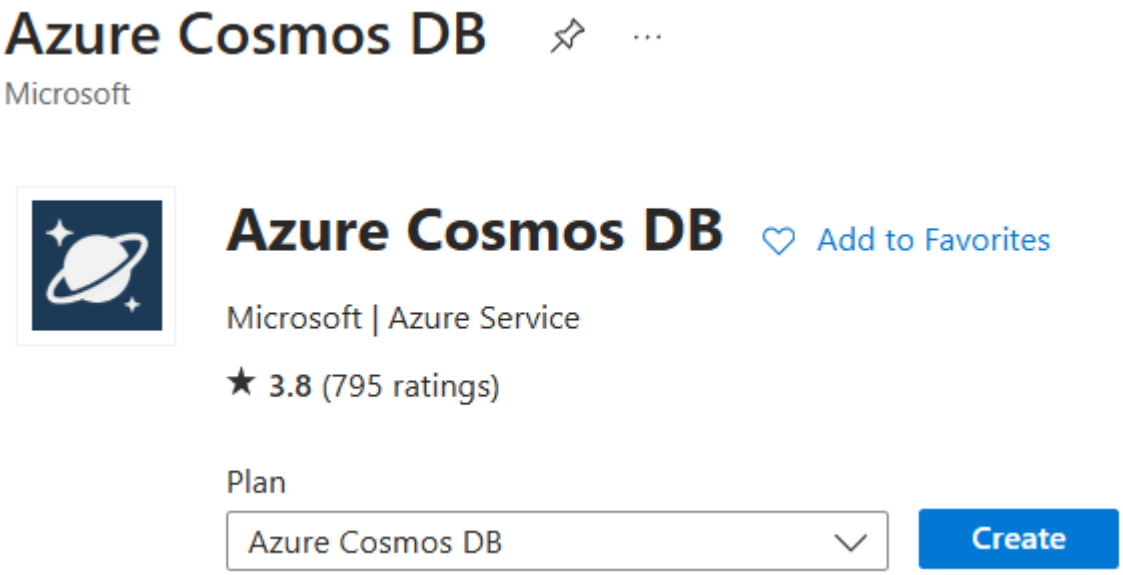
Conclusion

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.

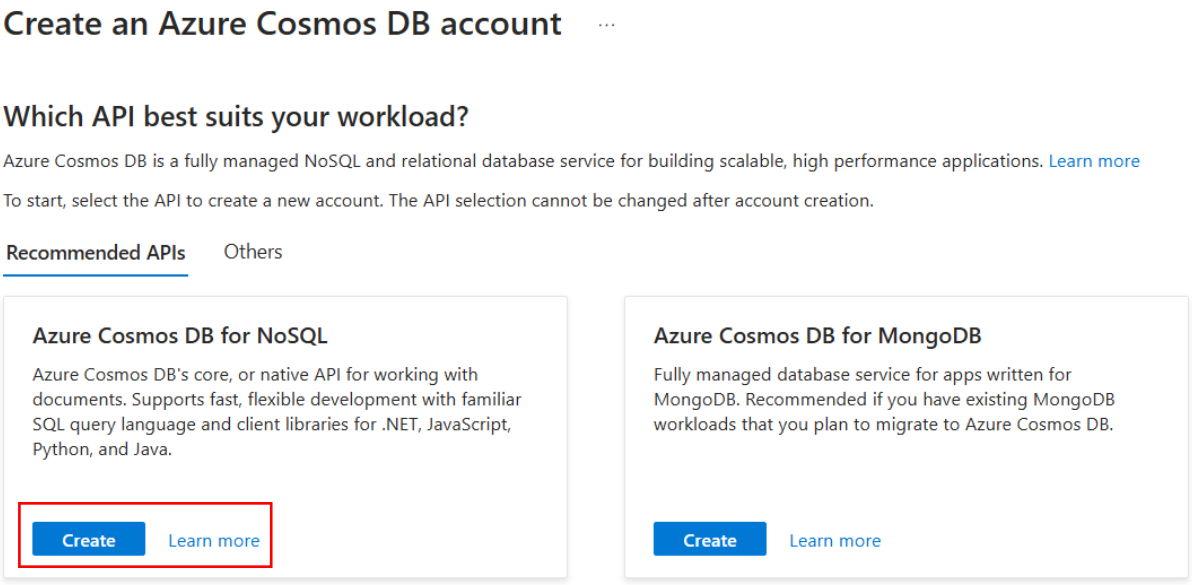
In this lab, we set up Azure Cosmos DB for NoSQL to store and manage JSON-based data. We begin by creating a Cosmos DB account in the Azure Portal, selecting NoSQL API, and configuring basic settings. After deployment, we use Data Explorer to create a database and a container with a partition key. We 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

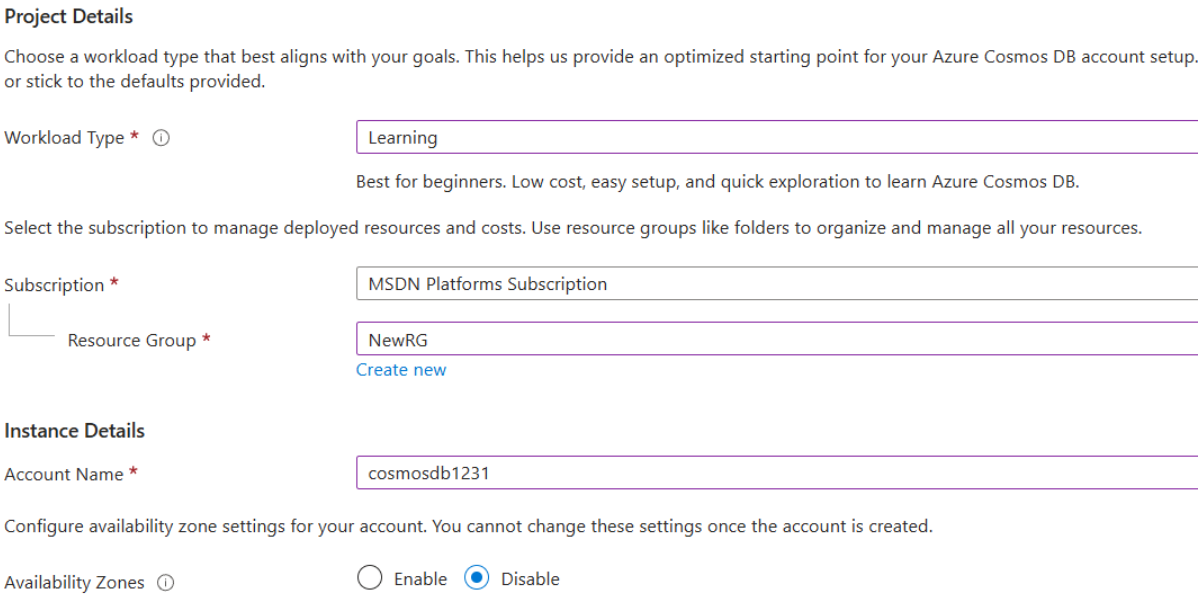
1. In this lab, we are going to work with Azure Cosmos DB. So, in the Azure Portal from the market place search for Azure Cosmos DB.



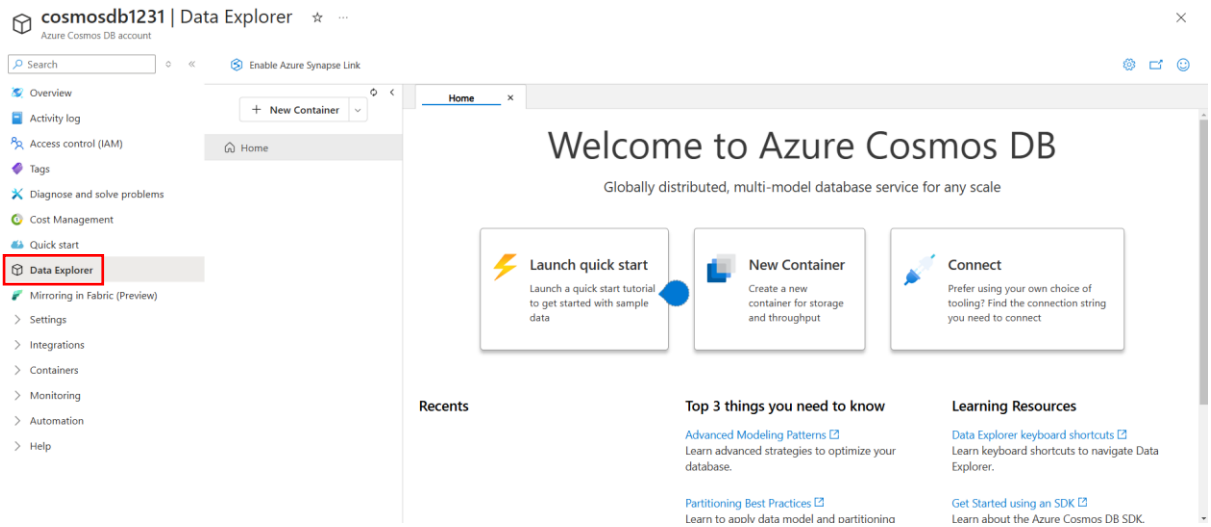
2. After clicking on Create button you will have two options to choose from. You have to choose Cosmos DB for NoSQL.



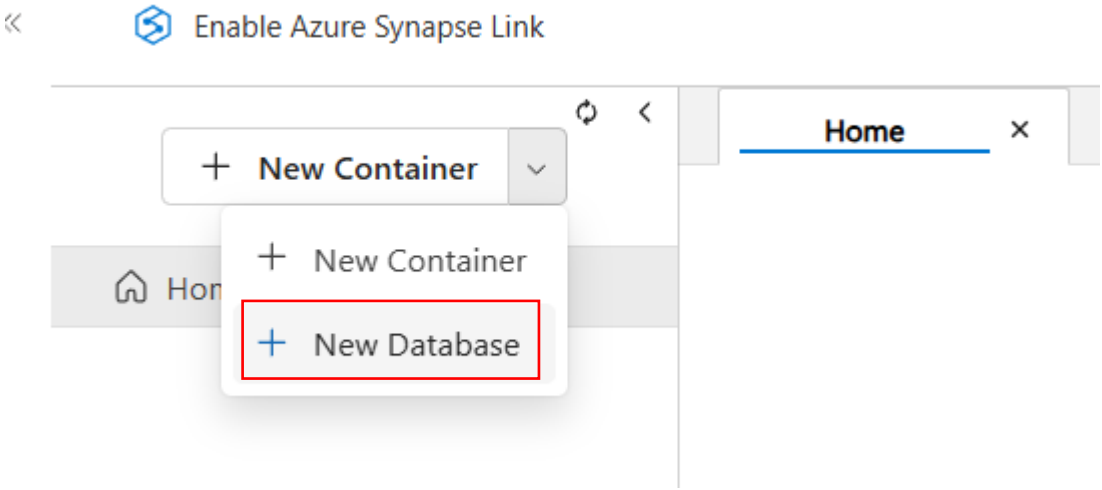
3. Now in the workload type choose learning, and select your subscription. Give a unique name to your Cosmos DB account. Then just leave every setting as it is and create your Cosmos DB.



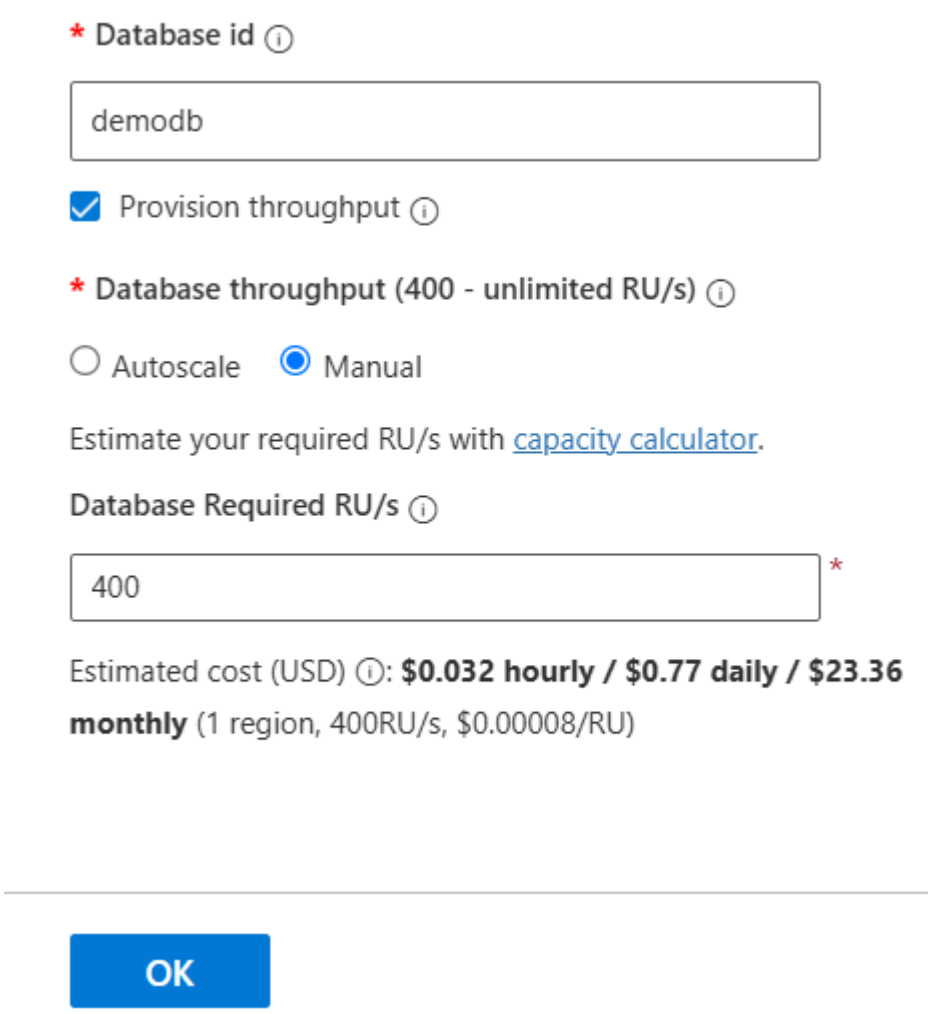
4. Once the deployment is complete, move to the resource.
5. Once you are inside the Cosmos DB open the data explorer. We can create now something known as a database within the account, and we can create something known as a container within the database.



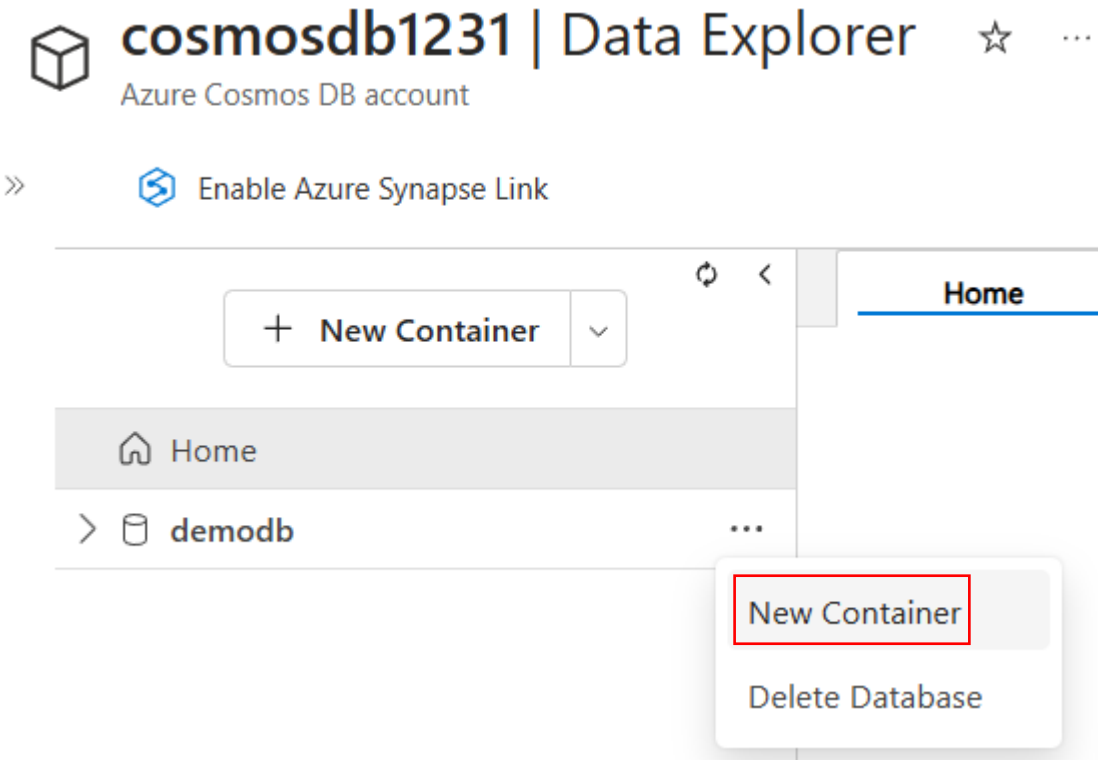
6. Now you need to click on the new database to create a new one.



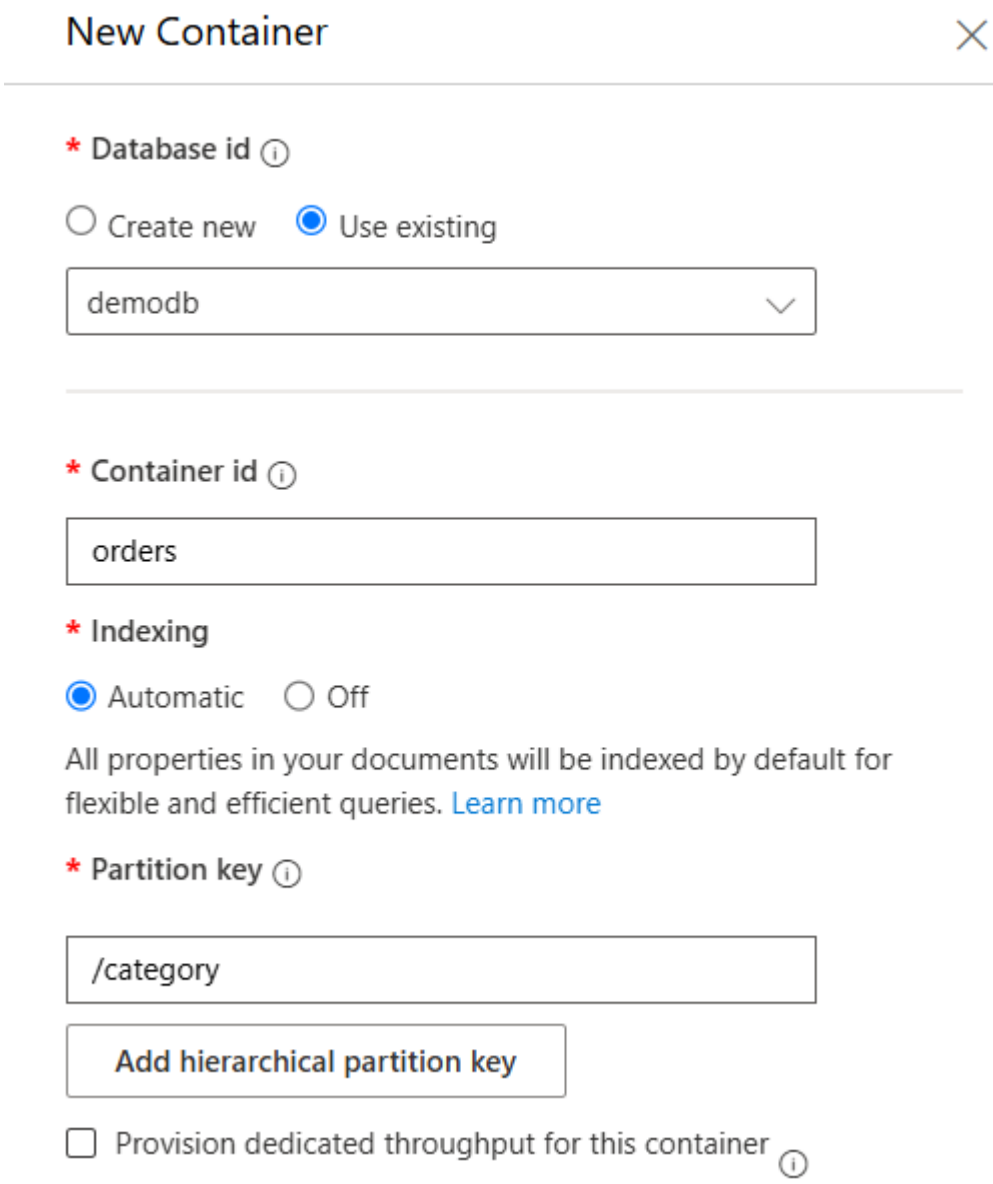
7. Give a name to your database ID and then in the provision throughput choose manual because we are not going to use much of the request units. Click on OK.



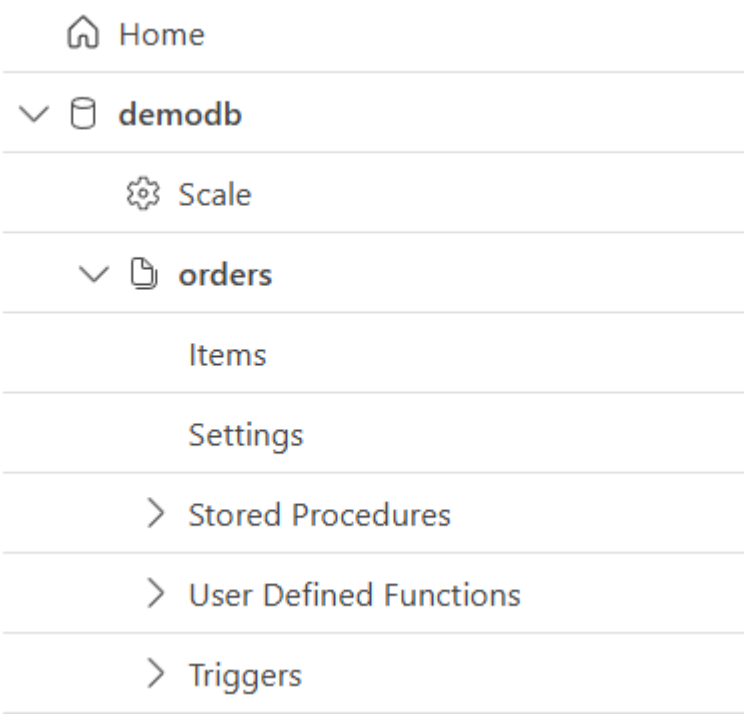
8. Within the database now we can create a new container.



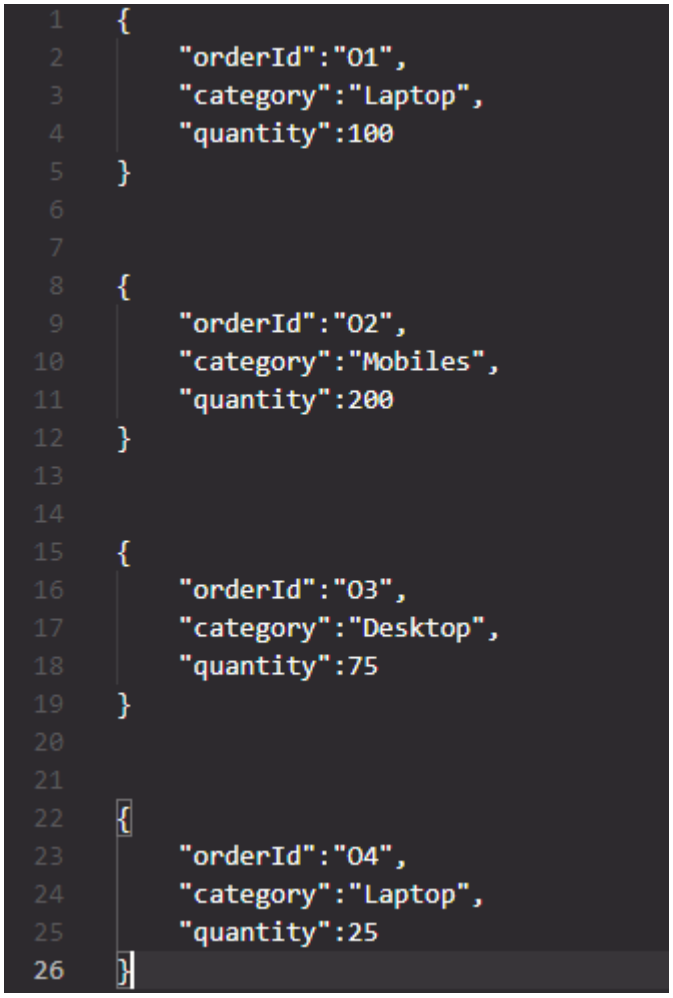
9. Here first we will give a container ID then the partition key. After that click on Ok.



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



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.



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


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


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


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


{

"orderId":"O1",

"category":"Laptop",

"quantity":100

}

{

"orderId":"O2",

"category":"Mobiles",

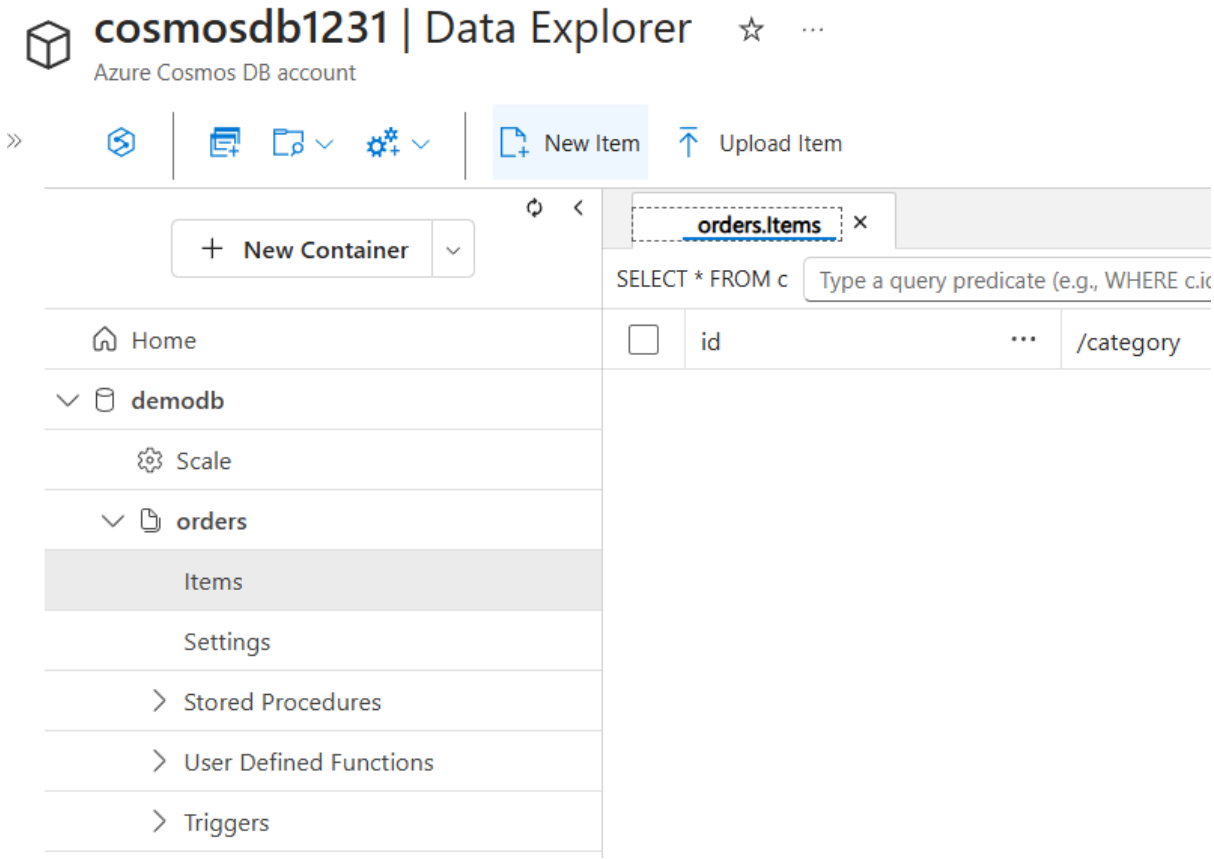
"quantity":200

}

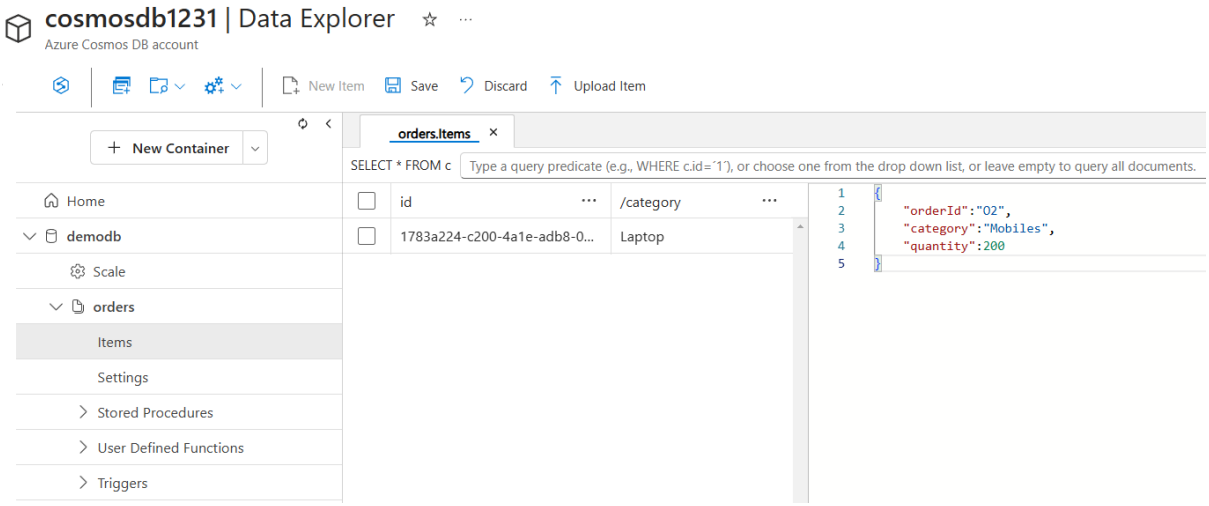
```
{
   "orderId":"O3",
   "category":"Desktop",
   "quantity":75
}
```

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

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



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



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