

Explore fundamentals of Azure Cosmos DB

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/explore-non-relational-data-stores-azure/1-introduction>

Introduction

Completed

Relational databases store data in relational tables, but sometimes the structure imposed by this model can be too rigid, and often leads to poor performance unless you spend time implementing detailed tuning. Other models, collectively known as *NoSQL* databases, exist. These models store data in other structures, such as documents, graphs, key-value stores, and column family stores.

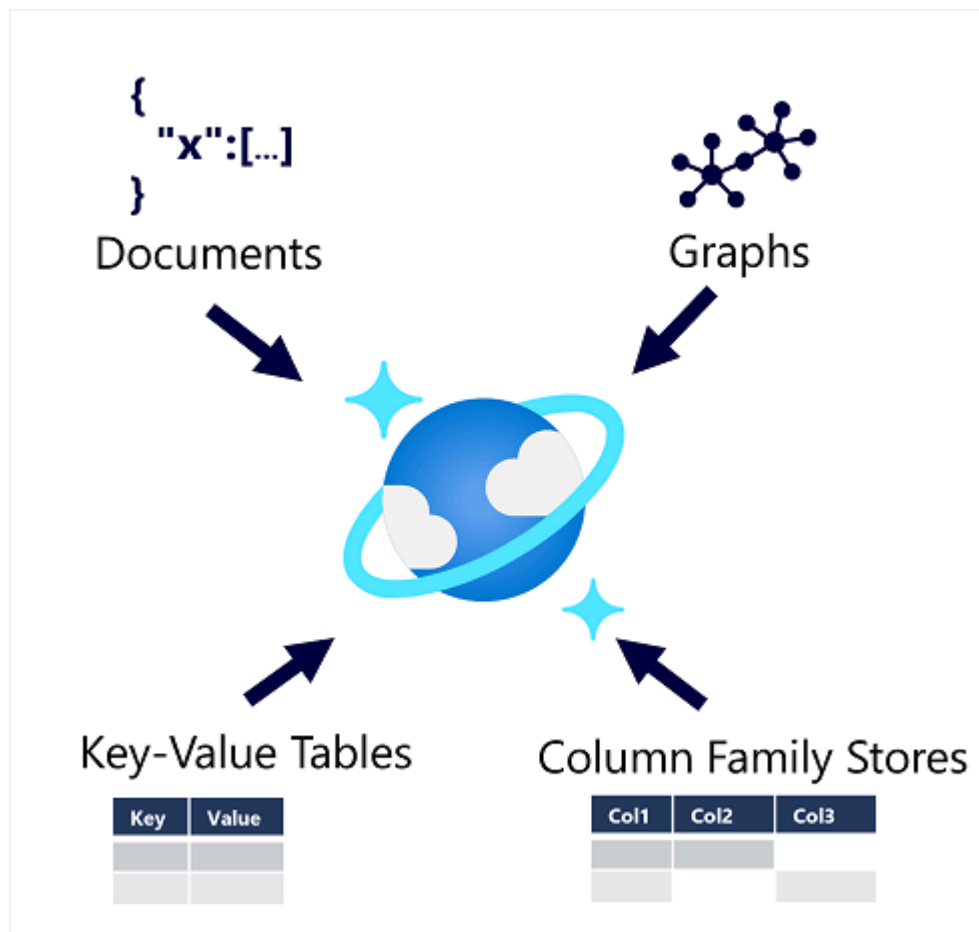
Azure Cosmos DB is a highly scalable cloud database service for NoSQL data.

2. Describe Azure Cosmos DB

<https://learn.microsoft.com/en-us/training/modules/explore-non-relational-data-stores-azure/2-describe-azure-cosmos-db>

Describe Azure Cosmos DB

Completed



Azure Cosmos DB supports multiple application programming interfaces (APIs) that enable developers to use the programming semantics of many common kinds of data store to work with data in a Cosmos DB database. The internal data structure is abstracted, enabling developers to use Cosmos DB to store and query data using APIs with which they're already familiar.

Note

An *API* is an *Application Programming Interface*. Database management systems (and other software frameworks) provide a set of APIs that developers can use to write programs that need to access data. The APIs vary for different database management systems.

Cosmos DB uses indexes and partitioning to provide fast read and write performance and can scale to massive volumes of data. You can enable multi-region writes, adding the Azure regions of your choice to your Cosmos DB account so that globally distributed users can each work with data in their local replica.

When to use Cosmos DB

Cosmos DB is a highly scalable database management system. Cosmos DB automatically allocates space in a container for your partitions, and each partition can grow up to 10 GB in size. Indexes are created and maintained automatically. There's virtually no administrative overhead.

Cosmos DB is a foundational service in Azure. Cosmos DB has been used by many of Microsoft's products for mission critical applications at global scale, including Skype, Xbox, Microsoft 365, Azure, and many others. Cosmos DB is highly suitable for the following scenarios:

- *IoT and telematics.* These systems typically ingest large amounts of data in frequent bursts of activity. Cosmos DB can accept and store this information quickly. The data can then be used by analytics services, such as Azure Machine Learning, Microsoft Fabric, and Power BI. Additionally, you can process the data in real-time using Azure Functions that are triggered as data arrives in the database.
- *Retail and marketing.* Microsoft uses Cosmos DB for its own e-commerce platforms that run as part of Windows Store and Xbox Live. It's also used in the retail industry for storing catalog data and for event sourcing in order processing pipelines.
- *Gaming.* The database tier is a crucial component of gaming applications. Modern games perform graphical processing on mobile/console clients, but rely on the cloud to deliver customized and personalized content like in-game stats, social media integration, and high-score leaderboards. Games often require single-millisecond latencies for reads and write to provide an engaging in-game experience. A game database needs to be fast and be able to handle massive spikes in request rates during new game launches and feature updates.
- *Web and mobile applications.* Azure Cosmos DB is commonly used within web and mobile applications, and is well suited for modeling social interactions, integrating with third-party services, and for building rich personalized experiences. The Cosmos DB SDKs can be used to build rich iOS and Android applications using the popular .NET MAUI framework.

For additional information about uses for Cosmos DB, read [Common Azure Cosmos DB use cases](#).

3. Identify Azure Cosmos DB APIs

<https://learn.microsoft.com/en-us/training/modules/explore-non-relational-data-stores-azure/3-cosmos-db-apis>

Identify Azure Cosmos DB APIs

Completed

Azure Cosmos DB is Microsoft's fully managed and serverless distributed database for applications of any size or scale, with support for both relational and non-relational workloads. Developers can build and migrate applications fast using their preferred open source database engines, including MongoDB and Apache Cassandra. When you provision a new Cosmos DB instance, you select the database engine that you want to use. The choice of engine depends on many factors including the type of data to be stored, the need to support existing applications, and the skills of the developers who work with the data store.

Azure Cosmos DB for NoSQL

Azure Cosmos DB for NoSQL is Microsoft's native non-relational service for working with the document data model. It manages data in JSON document format, and despite being a NoSQL data storage solution, uses SQL syntax to work with the data.

A SQL query for an Azure Cosmos DB database containing customer data might look similar to this:

```
SELECT *  
FROM customers c  
WHERE c.id = "joe@litware.com"
```

The result of this query consists of one or more JSON documents, as shown here:

```
{  
  "id": "joe@litware.com",  
  "name": "Joe Jones",  
  "address": {  
    "street": "1 Main St.",  
    "city": "Seattle"  
  }  
}
```

Azure Cosmos DB for MongoDB

MongoDB is a popular open source database in which data is stored in Binary JSON (BSON) format. Azure Cosmos DB for MongoDB enables developers to use MongoDB client libraries and code to work with data in Azure Cosmos DB.

MongoDB Query Language (MQL) uses a compact, object-oriented syntax in which developers use *objects* to call *methods*. For example, the following query uses the **find** method to query the **products** collection in the **db** object:

```
db.products.find({id: 123})
```

The results of this query consist of JSON documents, similar to this:

```
{  
  "id": 123,  
  "name": "Hammer",  
  "price": 2.99  
}
```

Azure Cosmos DB for Table

Azure Cosmos DB for Table is used to work with data in key-value tables, similar to Azure Table Storage. It offers greater scalability and performance than Azure Table Storage. For example, you might define a table named Customers like this:

PartitionKey	RowKey	Name	Email
1	123	Joe Jones	joe@litware.com
1	124	Samir Nadoy	samir@northwind.com

You can then use the Table API through one of the language-specific SDKs to make calls to your service endpoint to retrieve data from the table. For example, the following request returns the row containing the record for *Samir Nadoy* in the previous table:

```
https://endpoint/Customers(PartitionKey='1',RowKey='124')
```

Azure Cosmos DB for Apache Cassandra

Azure Cosmos DB for Apache Cassandra is compatible with Apache Cassandra, which is a popular open source database that uses a column-family storage structure. Column families are tables, similar to those in a relational database, with the exception that it's not mandatory for every row to have the same columns.

For example, you might create an **Employees** table like this:

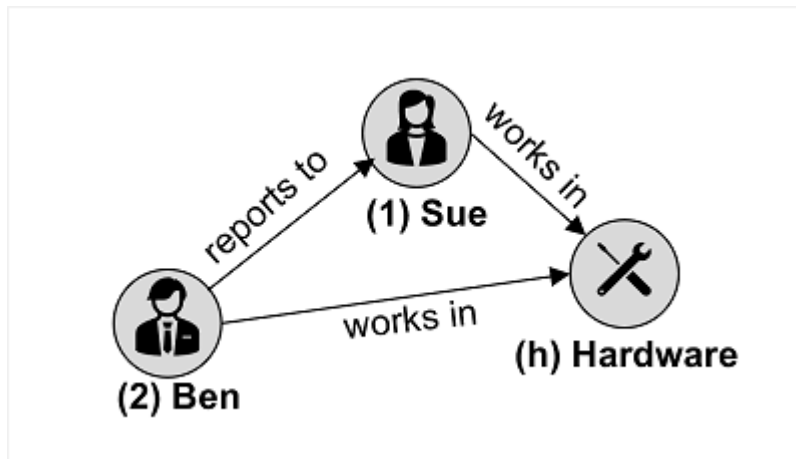
ID	Name	Manager
1	Sue Smith	
2	Ben Chan	Sue Smith

Cassandra supports a syntax based on SQL, so a client application could retrieve the record for *Ben Chan* like this:

```
SELECT * FROM Employees WHERE ID = 2
```

Azure Cosmos DB for Apache Gremlin

Azure Cosmos DB for Apache Gremlin is used with data in a graph structure; in which entities are defined as vertices that form nodes in connected graph. Nodes are connected by edges that represent relationships, like this:



The example in the image shows two kinds of vertex (employee and department) and edges that connect them (employee "Ben" reports to employee "Sue", and both employees work in the "Hardware" department).

Gremlin syntax includes functions to operate on vertices and edges, enabling you to insert, update, delete, and query data in the graph. For example, you could use the following code to add a new employee named *Alice* that reports to the employee with ID **1** (*Sue*)

```
g.addV('employee').property('id', '3').property('firstName', 'Alice')
g.V('3').addE('reports to').to(g.V('1'))
```

The following query returns all of the *employee* vertices, in order of ID.

```
g.V().hasLabel('employee').order().by('id')
```

4. Exercise: Explore Azure Cosmos DB

<https://learn.microsoft.com/en-us/training/modules/explore-non-relational-data-stores-azure/4-exercise-explore-cosmos-db>

Exercise: Explore Azure Cosmos DB

Completed

Now it's your opportunity to explore Azure Cosmos DB.

Note

To complete this lab, you will need an [Azure subscription](#) in which you have administrative access.

Launch the exercise and follow the instructions.

Launch Exercise

5. Module assessment

<https://learn.microsoft.com/en-us/training/modules/explore-non-relational-data-stores-azure/5-knowledge-check>

Module assessment

Completed

6. Summary

<https://learn.microsoft.com/en-us/training/modules/explore-non-relational-data-stores-azure/6-summary>

Summary

Completed

Azure Cosmos DB provides a global-scale database solution for non-relational data.

In this module, you learned how to:

- Describe key features and capabilities of Azure Cosmos DB
- Identify Azure Cosmos DB APIs
- Provision and use an Azure Cosmos DB instance

Next steps

Now that you've learned about Azure Cosmos DB for non-relational data storage, consider learning more about data-related workloads on Azure by pursuing a Microsoft certification in [Azure Data](#)

Fundamentals.