

Domain 3: Introduction to Microsoft Azure

Data non-relational data in Azure

Explore Azure Storage for non-relational data

1. Introduction

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

Introduction

Completed

Most software applications need to store data. Often data is stored in a relational database, in which the data is organized in related tables and managed by using Structured Query Language (SQL). However, many applications don't need the rigid structure of a relational database and rely on non-relational (often referred to as NoSQL) storage.

Azure Storage and Microsoft OneLake offer a range of options for storing data in the cloud. In this module, you explore the fundamental capabilities of Azure Storage and Microsoft OneLake, and learn how they're used to support applications that require non-relational data stores.

2. Explore Azure blob storage

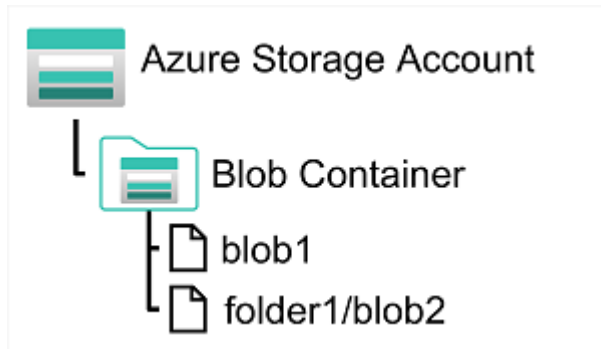
<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-non-relational-data-services-azure/2-azure-blob-storage>

Explore Azure blob storage

Completed

Azure Blob Storage is a service that enables you to store massive amounts of unstructured data as binary large objects, or *blobs*, in the cloud. Blobs are an efficient way to store data files in a format

that is optimized for cloud-based storage, and applications can read and write them by using the Azure blob storage API.



In an Azure storage account, you store blobs in *containers*. A container provides a convenient way of grouping related blobs together. You control who can read and write blobs inside a container at the container level.

Within a container, you can organize blobs in a hierarchy of virtual folders, similar to files in a file system on disk. However, by default, these folders are simply a way of using a "/" character in a blob name to organize the blobs into namespaces. The folders are purely virtual, and you can't perform folder-level operations to control access or perform bulk operations.

Azure Blob Storage supports three different types of blob:

- **Block blobs.** A block blob is handled as a set of blocks. Each block can vary in size, up to 4000 MiB. A block blob can contain up to 190.7 TiB (4000 MiB X 50,000 blocks). The block is the smallest amount of data that can be read or written as an individual unit. Block blobs are best used to store discrete, large, binary objects that change infrequently.
- **Page blobs.** A page blob is organized as a collection of fixed size 512-byte pages. A page blob is optimized to support random read and write operations; you can fetch and store data for a single page if necessary. A page blob can hold up to 8 TB of data. Azure uses page blobs to implement virtual disk storage for virtual machines.
- **Append blobs.** An append blob is a block blob optimized to support append operations. You can only add blocks to the end of an append blob; updating or deleting existing blocks isn't supported. Each block can vary in size, up to 4 MB. The maximum size of an append blob is just over 195 GB.

Blob storage provides four access tiers, which help to balance access latency and storage cost:

- The *Hot* tier is the default. You use this tier for blobs that are accessed frequently. The blob data is stored on high-performance media.
- The *Cool* tier has lower performance and incurs reduced storage charges compared to the Hot tier. Use the Cool tier for data that is accessed infrequently. Data should remain in the Cool tier for a minimum of 30 days to avoid early deletion penalties. It's common for newly created blobs to be accessed frequently initially, but less so as time passes. In these situations, you can create the blob in the Hot tier, but migrate it to the Cool tier later. You can migrate a blob from the Cool tier back to the Hot tier.

- The *Cold* tier is optimized for storing data that is rarely accessed or modified, but still requires fast retrieval. The Cold tier has lower storage costs and higher access costs compared to the Cool tier. Data should remain in the Cold tier for a minimum of 90 days to avoid early deletion penalties. Use this tier for data such as short-term backups, disaster recovery data, or large datasets that need cost-effective storage.
- The *Archive* tier provides the lowest storage cost, but with increased latency. The Archive tier is intended for historical data that mustn't be lost, but is required only rarely. Data should remain in the Archive tier for a minimum of 180 days to avoid early deletion penalties. Blobs in the Archive tier are effectively stored in an offline state. Typical reading latency for the Hot, Cool, and Cold tiers is a few milliseconds, but for the Archive tier, it can take up to 15 hours for the data to become available. To retrieve a blob from the Archive tier, you must change the access tier to Hot, Cool, or Cold. The blob will then be rehydrated. You can read the blob only when the rehydration process is complete.

You can create lifecycle management policies for blobs in a storage account. A lifecycle management policy can automatically move a blob from Hot to Cool, then to Cold, and finally to the Archive tier, as it ages and is used less frequently (policy is based on the number of days since modification). A lifecycle management policy can also arrange to delete outdated blobs.

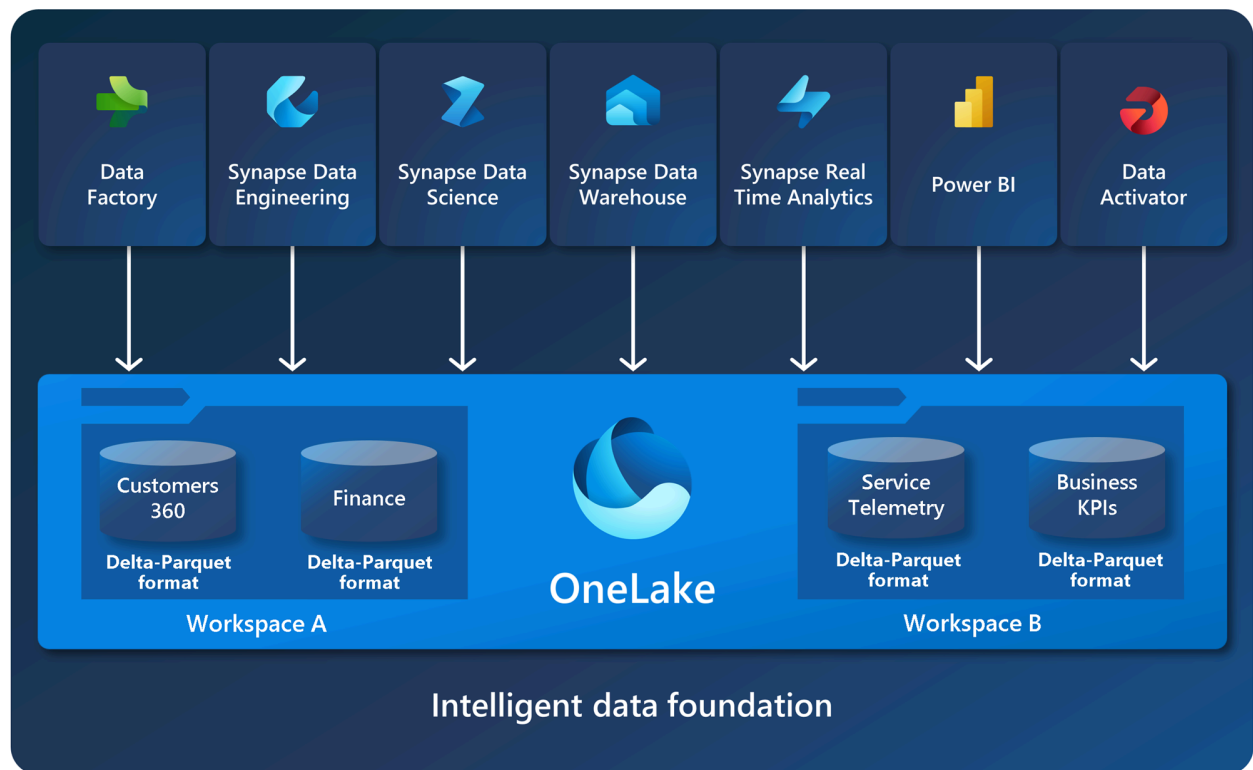
3. Explore Microsoft OneLake in Fabric

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-non-relational-data-services-azure/3a-one-lake>

Explore Microsoft OneLake in Fabric

Completed

Microsoft Fabric automatically provisions OneLake, built upon Azure Data Lake Gen 2.



OneLake is a single, unified, logical data lake designed for your entire organization. OneLake comes automatically with every Microsoft Fabric tenant and serves as the central repository for all your analytics data. Whether structured or unstructured, OneLake supports any type of file and allows you to use the same data across multiple analytical engines without data movement or duplication.

Key Benefits of OneLake

- **Organization-wide data lake** Before OneLake, creating multiple data lakes for different business groups was common. Now, OneLake provides a collaborative solution, ensuring that your entire organization shares a single data lake.
- **Distributed ownership and collaboration** Within a tenant, you can create workspaces, enabling different parts of your organization to manage their data items. This distributed ownership promotes collaboration while maintaining governance boundaries.
- **Open and Compatible** Built on Azure Data Lake Storage (ADLS) Gen2, OneLake stores data in Delta Parquet format. It supports existing ADLS Gen2 APIs and SDKs, making it compatible with your current applications.
- **Easy to navigate** It's straightforward to navigate OneLake data from Windows using [OneLake file explorer](#).

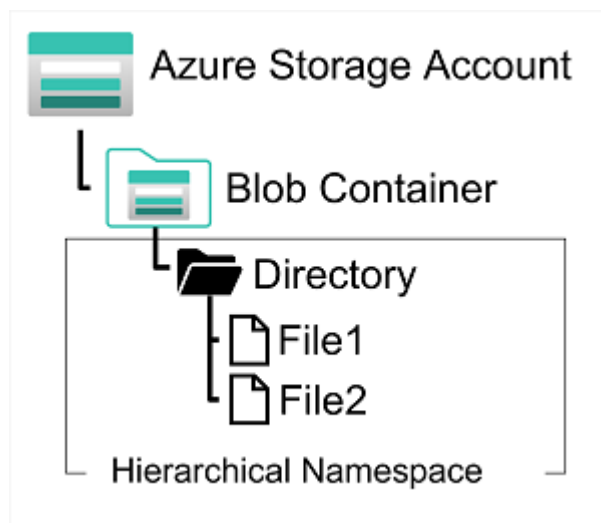
For more information, see [OneLake, the OneDrive for data](#).

4. Explore Azure Data Lake Storage Gen2

Explore Azure Data Lake Storage Gen2

Completed

Azure Data Lake Store (Gen1) is a separate service for hierarchical data storage for analytical data lakes, often used by so-called *big data* analytical solutions that work with structured, semi-structured, and unstructured data stored in files. Azure Data Lake Storage Gen2 is a newer version of this service that is integrated into Azure Storage; enabling you to take advantage of the scalability of blob storage and the cost-control of storage tiers, combined with the hierarchical file system capabilities and compatibility with major analytics systems of Azure Data Lake Store.



Systems like Azure Databricks can mount a distributed file system hosted in Azure Data Lake Store Gen2 and use it to process huge volumes of data. Microsoft Fabric tenants automatically provision OneLake, built on top of Azure Data Lake Storage Gen2.

To create an Azure Data Lake Store Gen2 files system, you must enable the **Hierarchical Namespace** option of an Azure Storage account. You can do this when initially creating the storage account, or you can upgrade an existing Azure Storage account to support Data Lake Gen2. Be aware however that upgrading is a one-way process – after upgrading a storage account to support a hierarchical namespace for blob storage, you can't revert it to a flat namespace.

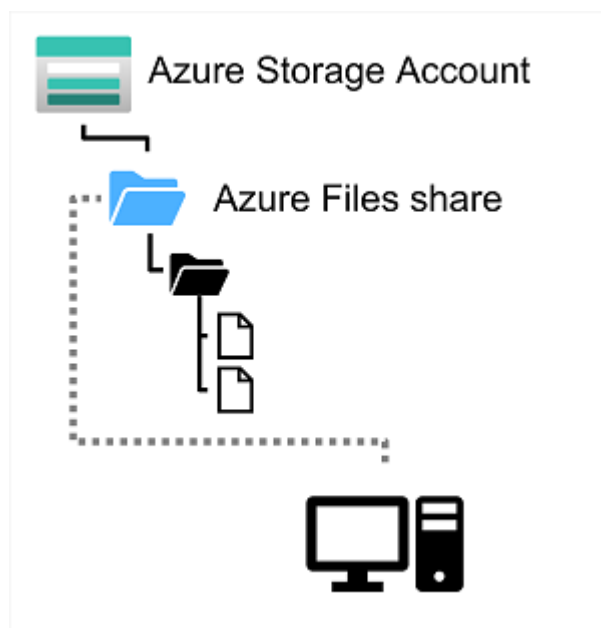
5. Explore Azure Files

Explore Azure Files

Completed

Many on-premises systems comprising a network of in-house computers make use of file shares. A file share enables you to store a file on one computer, and grant access to that file to users and applications running on other computers. This strategy can work well for computers in the same local area network, but doesn't scale well as the number of users increases, or if users are located at different sites.

Azure Files is essentially a way to create cloud-based network shares, such as you typically find in on-premises organizations to make documents and other files available to multiple users. By hosting file shares in Azure, organizations can eliminate hardware costs and maintenance overhead, and benefit from high availability and scalable cloud storage for files.



You create Azure File storage in a storage account. Azure Files enables you to share up to 100 TB of data in a single storage account. This data can be distributed across any number of file shares in the account. The maximum size of a single file is 4 TiB, but you can set quotas to limit the size of each share below this figure. Currently, Azure File Storage supports up to 2000 concurrent connections per shared file.

After you've created a storage account, you can upload files to Azure File Storage using the Azure portal, or tools such as the *AzCopy* utility. You can also use the Azure File Sync service to synchronize locally cached copies of shared files with the data in Azure File Storage.

Azure File Storage offers two performance tiers. The *Standard* tier uses hard disk-based hardware in a datacenter, and the *Premium* tier uses solid-state disks. The *Premium* tier offers greater throughput, but is charged at a higher rate.

Azure Files supports two common network file sharing protocols:

- *Server Message Block* (SMB) file sharing is commonly used across multiple operating systems (Windows, Linux, macOS).
- *Network File System* (NFS) shares are used by some Linux and macOS versions. To create an NFS share, you must use a premium tier storage account and create and configure a virtual network through which access to the share can be controlled.

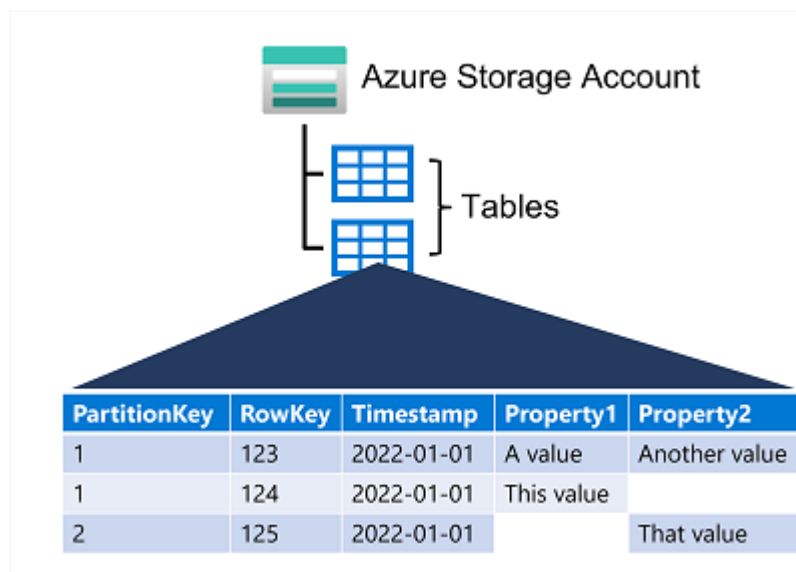
6. Explore Azure Tables

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-non-relational-data-services-azure/5-azure-tables>

Explore Azure Tables

Completed

Azure Table Storage is a NoSQL storage solution that makes use of tables containing *key/value* data items. Each item is represented by a row that contains columns for the data fields that need to be stored.



However, don't be misled into thinking that an Azure Table Storage table is like a table in a relational database. An Azure Table enables you to store semi-structured data. All rows in a table must have a

unique key (composed of a partition key and a row key), and when you modify data in a table, a *timestamp* column records the date and time the modification was made; but other than that, the columns in each row can vary. Azure Table Storage tables have no concept of foreign keys, relationships, stored procedures, views, or other objects you might find in a relational database. Data in Azure Table storage is usually denormalized, with each row holding the entire data for a logical entity. For example, a table holding customer information might store the first name, last name, one or more telephone numbers, and one or more addresses for each customer. The number of fields in each row can be different, depending on the number of telephone numbers and addresses for each customer, and the details recorded for each address. In a relational database, this information would be split across multiple rows in several tables.

To help ensure fast access, Azure Table Storage splits a table into partitions. Partitioning is a mechanism for grouping related rows, based on a common property or partition key. Rows that share the same partition key will be stored together. Partitioning not only helps to organize data, it can also improve scalability and performance in the following ways:

- Partitions are independent from each other, and can grow or shrink as rows are added to, or removed from, a partition. A table can contain any number of partitions.
- When you search for data, you can include the partition key in the search criteria. This helps to narrow down the volume of data to be examined, and improves performance by reducing the amount of I/O (input and output operations, or *reads* and *writes*) needed to locate the data.

The key in an Azure Table Storage table comprises two elements; the partition key that identifies the partition containing the row, and a row key that is unique to each row in the same partition. Items in the same partition are stored in row key order. If an application adds a new row to a table, Azure ensures that the row is placed in the correct position in the table. This scheme enables an application to quickly perform *point* queries that identify a single row, and *range* queries that fetch a contiguous block of rows in a partition.

7. Exercise: Explore Azure Storage

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-non-relational-data-services-azure/6-exercise-azure-storage>

Exercise: Explore Azure Storage

Completed

Now it's your opportunity to explore Azure Storage.

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

8. Module assessment

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-non-relational-data-services-azure/7-knowledge-check>

Module assessment

Completed

9. Summary

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-non-relational-data-services-azure/8-summary>

Summary

Completed

Azure Storage is a key service in Microsoft Azure, and enables a wide range of data storage scenarios and solutions.

In this module, you learned how to:

- Describe features and capabilities of Azure blob storage
- Describe features and capabilities of Azure Data Lake Gen2
- Describe features and capabilities of Microsoft OneLake
- Describe features and capabilities of Azure file storage
- Describe features and capabilities of Azure table storage
- Provision and use an Azure Storage account

Next steps

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

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.

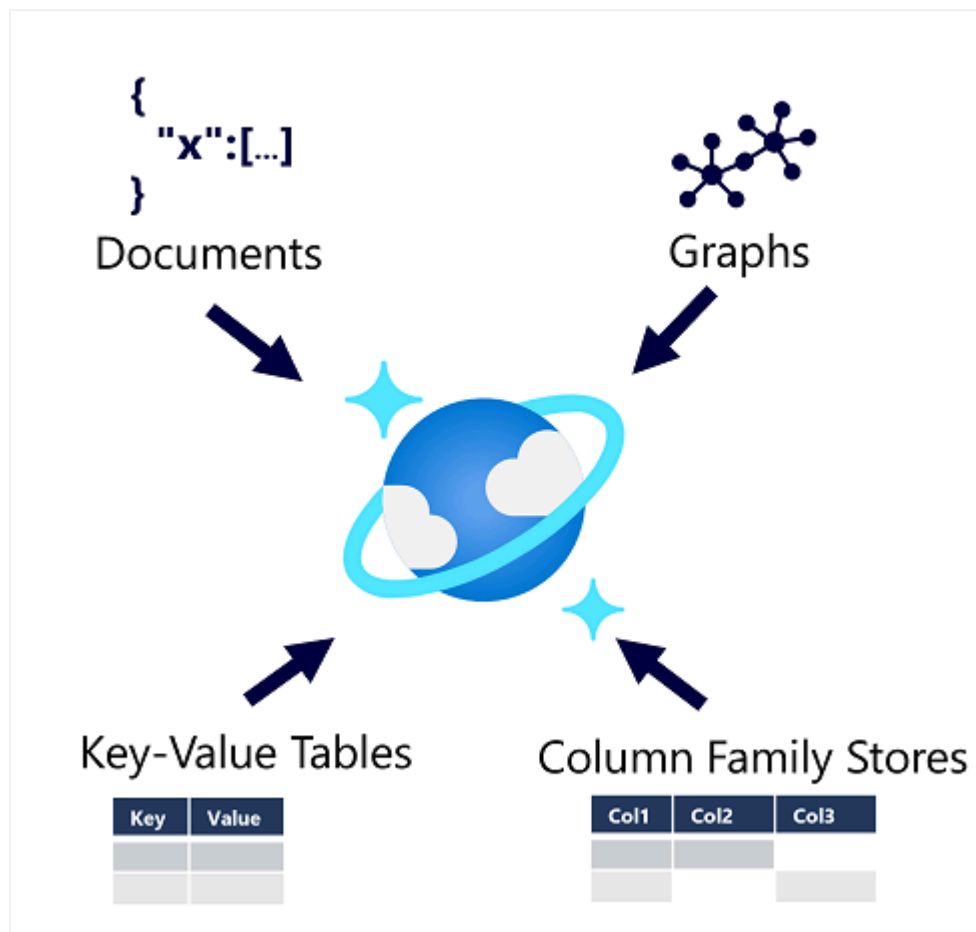
Azure Cosmos DB is a highly scalable cloud database service for NoSQL data.

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:

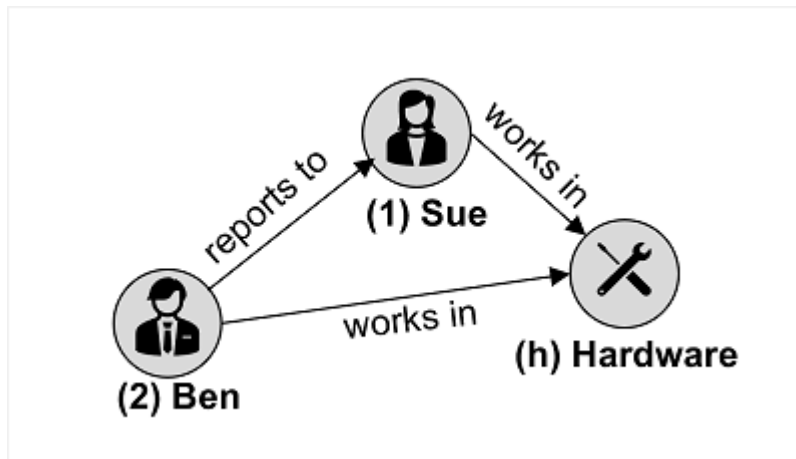
ID	Name	Manager
1	Sue Smith	
2	Ben Chan	Sue Smith

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.