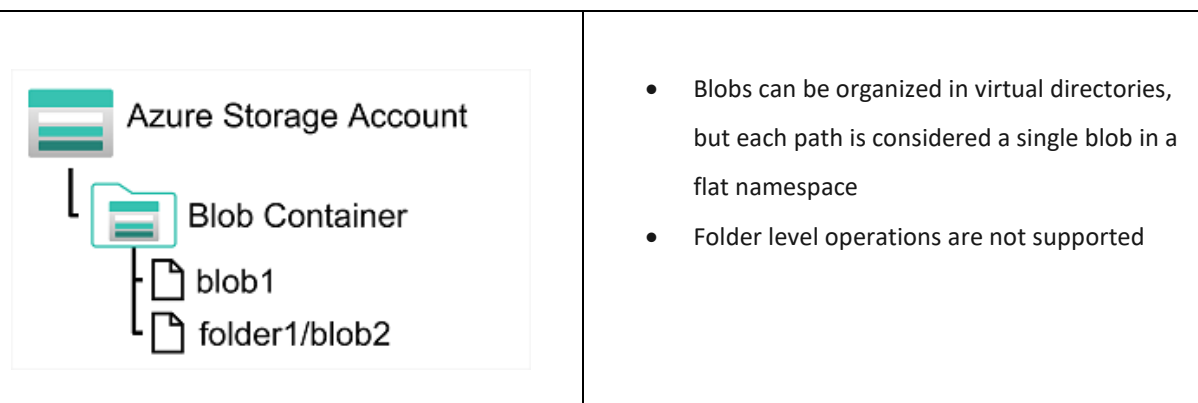


Azure Storage Fundamentals

- Azure Storage is a **PaaS service** that you can use to store both **unstructured** and **partially structured** data.
- **Azure Storage is massively scalable and elastic:** It can store and process **hundreds of terabytes of data** to support the big data scenarios required by scientific, financial analysis, and media applications. Or you can store the **small amounts of data** required for a small business website.

Azure Blob Storage:

- 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.



Types of Blobs:

1. **Block blobs** are optimized for streaming (**sequential access**) and for uploads and downloads, and are a good choice for storing documents, media files, backups etc. Azure divides data into smaller blocks of up to 100 megabytes (MB) in size, which subsequently upload or download in parallel.
Maximum size of a block in a block blob: 4000 MiB
Maximum size of a block blob: 50,000 X 4000 MiB (approximately 190.7 TiB)
2. **Append blobs:** Append blobs are similar to block blobs, but are optimized for append operations. This works best with **logging and auditing** activities. Updating or deleting of existing blocks is not supported.
Maximum size of a block in an append blob 4 MiB
Maximum size of an append blob 50,000 x 4 MiB (approximately 195 GiB)
3. **Page blobs** are optimized for **random read/write** operations and provide the ability to write to a range of bytes in a blob. Blobs are accessed as pages, each of which is up to **512 bytes** in size. Each Page blob can be up to **8TB** each. Is best suited for **virtual machine disks**.

Access Tiers:

1. **Hot tier:** is the default. You use this tier for blobs that are accessed frequently. The blob data is stored on high-performance media.
2. **The Cool tier:** This tier has lower performance and incurs reduced storage charges compared to the Hot tier.

Use the Cool tier for data that is accessed infrequently.

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.

3. **The Archive tier:** This 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.

Blobs in the Archive tier are effectively stored in an offline state.

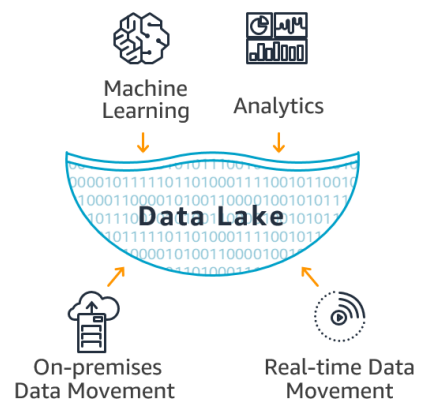
It can take hours for the data to become available and accessing that data is more expensive

To retrieve a blob from the Archive tier, you must change the access tier to Hot or Cool. The blob will then be rehydrated. You can read the blob only when the rehydration process is complete.

It is available at level of an individual blob only, not at the storage account level. Only block blobs and append blobs can be archived.

Azure DataLake Gen2:

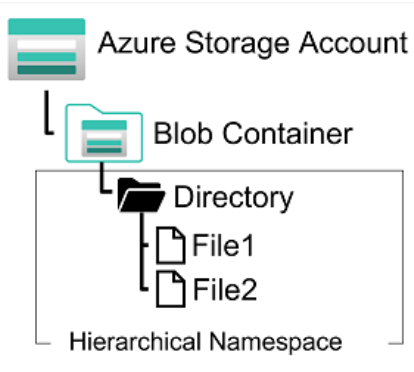
- Data lake storage is **storage solution** for Big Data
- **Centralized repository** for all enterprise data
- Data Stored in **natural** format.
- Built on **HDFS** Standard



Ref:

Azure data lake storage Gen 2 Features:

- Distributed file system built on Blob Storage
- Combines Azure Data Lake Store Gen 1 with Azure Blob Storage for large-scale file storage and analytics

| | |
|--|--|
| <ul style="list-style-type: none"> • Enables file and directory level access control and management • Compatible with common large scale analytical systems. • Systems like Hadoop in Azure HDInsight, Azure Databricks, and Azure Synapse Analytics can mount a distributed file system hosted in Azure Data Lake Store Gen 2 and use it to process huge volumes of data. | |
| <p>Multi modal storage service:</p>  <p>The diagram illustrates the Multi-modal storage service architecture. It shows an 'Azure Storage Account' at the top, which contains a 'Blob Container'. Inside the 'Blob Container' is a 'Directory', which in turn contains two files, 'File1' and 'File2'. The entire structure is labeled as a 'Hierarchical Namespace'.</p> | <ul style="list-style-type: none"> • Built on Azure Blob Storage. • Enable hierarchical namespace (HNS) property for general purpose V2, storage account <p>Features from Azure Data Lake Storage Gen1, such as file system semantics, directory, and file level security, capabilities for analytics workloads and scale are combined with low-cost, tiered storage (Hot/Cool/Archive), high availability/disaster recovery (RA-GRS) capabilities from Azure Blob storage.</p> |

Azure Files:

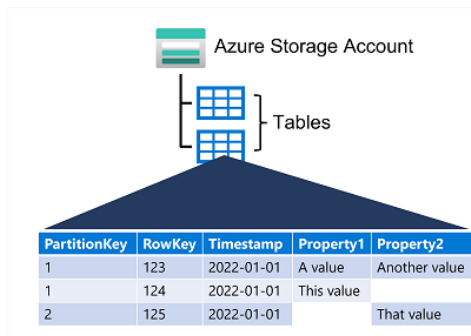
- Azure File storage is a service that offers file shares in the cloud.
- A file share enables you to store a file on one computer, and grant access to that file to users and applications running on other computers.
- Microsoft Azure virtual machines can share file data across application components via mounted shares, and on-premises applications can access file data in a share via the File storage API.
- With Azure File storage, you can migrate **legacy applications** that rely on file shares to Azure quickly and without costly rewrites.
- It enables you to share up to 100 TB of data in a single storage account Which can be distributed across any number of file shares in the account. Currently, Azure File Storage supports up to 2000 concurrent connections per shared file.
- The maximum size of a single file is 1 TB, but you can set quotas to limit the size of each share

Supported 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.

Azure Tables:

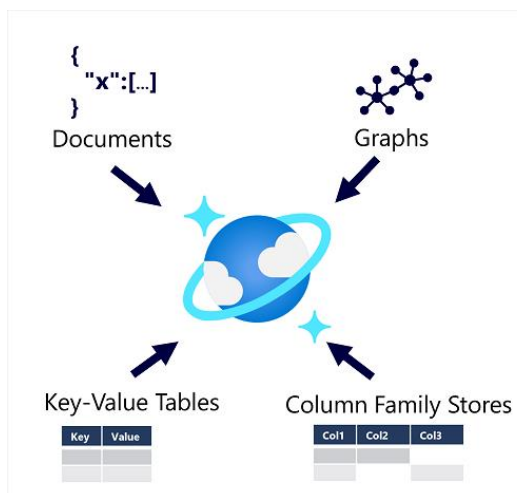
- Azure Table Storage is a NoSQL storage solution that makes use of tables containing *key/value* data items.



- Number of columns in each row can vary.
- Data in Azure Table storage is usually denormalized, with each row holding the entire data for a logical entity
- Azure Table Storage tables have no concept of foreign keys, relationships, stored procedures, views, or other objects
- Each row in a table must have a unique key (composed of a partition key and a row key)
- Rows that share the same partition key will be stored together in row key order
- Partitioning not only helps to organize data, but it can also improve scalability and performance

CosmosDB Fundamentals

- Azure Cosmos DB is **Microsoft's globally distributed, multi-model/multi-api NoSQL database**.
- 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.



CosmosDB API:

1. **SQL API (Core API)**: A schema-less JSON database engine with rich SQL querying capabilities.

2. **MongoDB API:** It's based on BSON Document format. A massively scalable *MongoDB-as-a-Service* powered by Azure Cosmos DB platform. Compatible with existing MongoDB libraries, drivers, tools, and applications.
3. **Table API:** A key-value database service built to provide premium capabilities (for example, automatic indexing, guaranteed low latency, global distribution) to existing Azure Table storage applications without making any app changes.
4. **Gremlin API:** A fully managed, horizontally scalable graph database service that makes it easy to build and run applications that work with highly connected datasets supporting Open Gremlin APIs. It stores entities which are called Nodes and Edges.
5. **Cassandra API:** A globally distributed Cassandra-as-a-Service powered by Azure Cosmos DB platform. Compatible with existing [Apache Cassandra](#) libraries, drivers, tools, and applications.
6. 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.

Core (SQL) API

- Native API for Cosmos DB
- SQL queries based on JSON documents

```

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

```

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

MongoDB API

- Compatibility with MongoDB
a popular open source document-based database

```

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

```

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

Table API

- Key-value storage API
- Compatible with Azure Table Storage, but with higher performance and scalability

| PartitionKey | RowKey | Name |
|--------------|--------|----------------|
| 1 | 123 | Joe Jones |
| 1 | 124 | Sammy Sprocket |

Cassandra API

- Compatibility with Apache Cassandra
a popular open source column-family database

```

SELECT *
FROM store.employee
WHERE dept='Hardware'
  
```

| id | name | dept | manager |
|----|-----------|----------|-----------|
| 1 | Sue Smith | Hardware | |
| 2 | Ben Chan | Hardware | Sue Smith |

Gremlin API

- Used to work with *graph* data
- Entity nodes (*vertices*) are connected via relationships (*edges*)

```

graph LR
    Ben["(2) Ben"] -- "reports to" --> Sue["(1) Sue"]
    Sue -- "works in" --> Hardware["(h) Hardware"]
    Ben -- "works in" --> Hardware
  
```

- 1. Retail Industry (Catalog Data)**
Catalog data are user accounts, product catalogs etc
- 2. IOT and telematics:**
Cosmos DB can accept and store this information quickly. The data can then be used by analytics services, such as Azure Machine Learning, Azure HDInsight, and Power BI.
- 3. Gaming:**

Modern Games, rely on the cloud to deliver customized and personalized content like in-game stats, social media integration, and high-score leader boards. Games often require single-millisecond latencies for reads and write to provide an engaging in-game experience.

Demo:

Refer: [Exercise: Explore Azure Cosmos DB - Learn | Microsoft Docs](#)

Lab1: Create CosmosDB Account

1. Azure Portal → +Create a resource → Azure Cosmos DB → Create → Select Core (SQL) Core → create

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

Subscription *

Resource Group * [Create new](#)

Instance Details

Account Name *

Location *

Capacity mode ☒ Provisioned throughput ☐ Serverless
[Learn more about capacity mode](#)

With Azure Cosmos DB free tier, you will get the first 1000 RU/s and 25 GB of storage for free in an account. You can enable free tier on up to one account per subscription. Estimated \$64/month discount per account.

Apply Free Tier Discount ☒ Apply ☐ Do Not Apply

Limit total account throughput ☒ Limit the total amount of throughput that can be provisioned on this account
ⓘ This limit will prevent unexpected charges related to provisioned throughput. You can update or remove this limit after your account is created.

→ Keep all default options → Review + Create

Sample Families Document

2. Add Sample data: Click on New Document

```
{
  "familyName": "Smith",
  "address": {
    "addressLine": "123 Main Street",
    "city": "Chicago",
    "state": "IL",
    "zipCode": "60601"
  },
  "parents": [
    "Peter",
    "Alice"
  ],
  "kids": [
    "Adam",
    "Jacqueline",
    "Joshua"
  ]
}
```

```
{
  "familyName": "Jones",
  "address": {
    "addressLine": "456 Harbor Boulevard",
    "city": "Chicago",
    "state": "IL",
    "zipCode": "60603"
  },
  "parents": [
    "David",
    "Diana"
  ],
  "kids": [
    "Evan"
  ],
  "pets": [
    "Lint"
  ]
}
```

Query Document:

```
SELECT * FROM c
WHERE STARTSWITH(c.address.addressline,123)
```

```
SELECT * FROM c
WHERE c.address.zipcode="60601"
```

```
SELECT * FROM c
WHERE ARRAY_LENGTH(c.kids)>2
```

```
SELECT * FROM c
WHERE IS_DEFINED(c.pets)
```

```
SELECT * FROM c
WHERE c.address.city="Chicago"
```

Refer: <https://docs.microsoft.com/en-us/azure/cosmos-db/sql-query-getting-started>