Azure Blob storage

Blob Storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that doesn't adhere to a particular data model or definition, such as text or binary data.

Blob Storage is designed for:

- Serving images or documents directly to a browser.
- Storing files for distributed access.
- Streaming video and audio.
- Writing to log files.
- Storing data for backup and restore, disaster recovery, and archiving.
- Storing data for analysis by an on-premises or Azure-hosted service.

Users or client applications can access objects in Blob Storage via HTTP/HTTPS, from anywhere in the world.
Objects in Blob Storage are accessible via the Azure Storage REST API, Azure PowerShell, Azure CLI, or an Azure Storage client library.
Client libraries are available for different languages, including:

- .NET
- Java
- Node.js
- Python
- Go

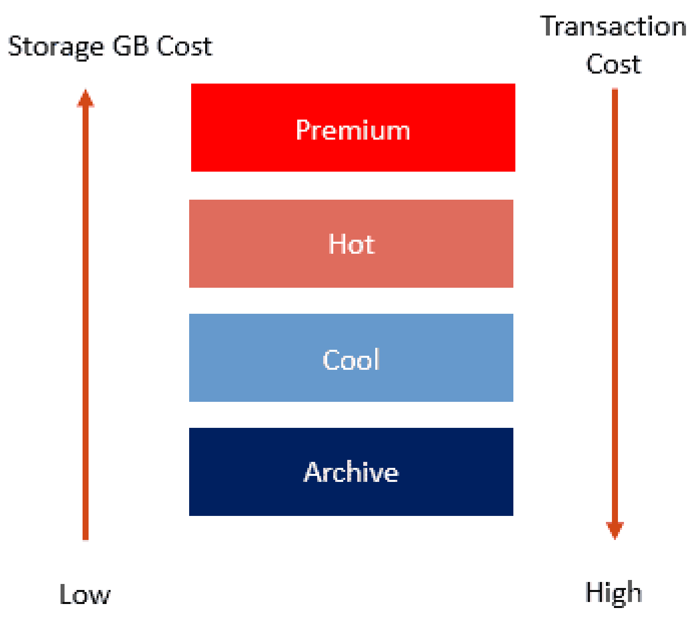
Clients can also securely connect to Blob Storage by using SSH File Transfer Protocol (SFTP) and mount Blob Storage containers by using the Network File System (NFS) 3.0 protocol.

Blob Storage supports Azure Data Lake Storage Gen2, Microsoft's enterprise big data analytics solution for the cloud. Azure Data Lake Storage Gen2 offers a hierarchical file system as well as the advantages of Blob Storage, including:

- Low-cost, tiered storage
- High availability
- Strong consistency
- Disaster recovery capabilities

Access tiers for blob data

Data stored in the cloud grows at an exponential pace. To manage costs for your expanding storage needs, it can be helpful to organize your data based on how frequently it will be accessed and how long it will be retained.



Azure Storage access tiers include:
Hot tier - An online tier optimized for storing data that is accessed or modified frequently. The hot tier has the highest storage costs, but the lowest access costs.
Example usage scenarios for the hot tier include:

- Data that's in active use or data that you expect will require frequent reads and writes.
- Data that's staged for processing and eventual migration to the cool access tier.

Usage scenarios for the cool and cold access tiers include:

- Short-term data backup and disaster recovery.
- Older data sets that aren't used frequently but are expected to be available for immediate access.
- Large data sets that need to be stored in a cost-effective way while other data is being gathered for processing.

Cool tier - An online tier optimized for storing data that is infrequently accessed or modified. Data in the cool tier should be stored for a minimum of 30 days. The cool tier has lower storage costs and higher access costs compared to the hot tier.

Cold tier - An online tier optimized for storing data that is rarely accessed or modified, but still requires fast retrieval. Data in the cold tier should be stored for a minimum of 90 days. The cold tier has lower storage costs and higher access costs compared to the cool tier.

Archive tier - An offline tier optimized for storing data that is rarely accessed, and that has flexible latency requirements, on the order of hours. Data in the archive tier should be stored for a minimum of 180 days.
Example usage scenarios for the archive access tier include:

- Long-term backup, secondary backup, and archival datasets
- Original (raw) data that must be preserved, even after it has been processed into final usable form

- Compliance and archival data that needs to be stored for a long time and is hardly ever accessed

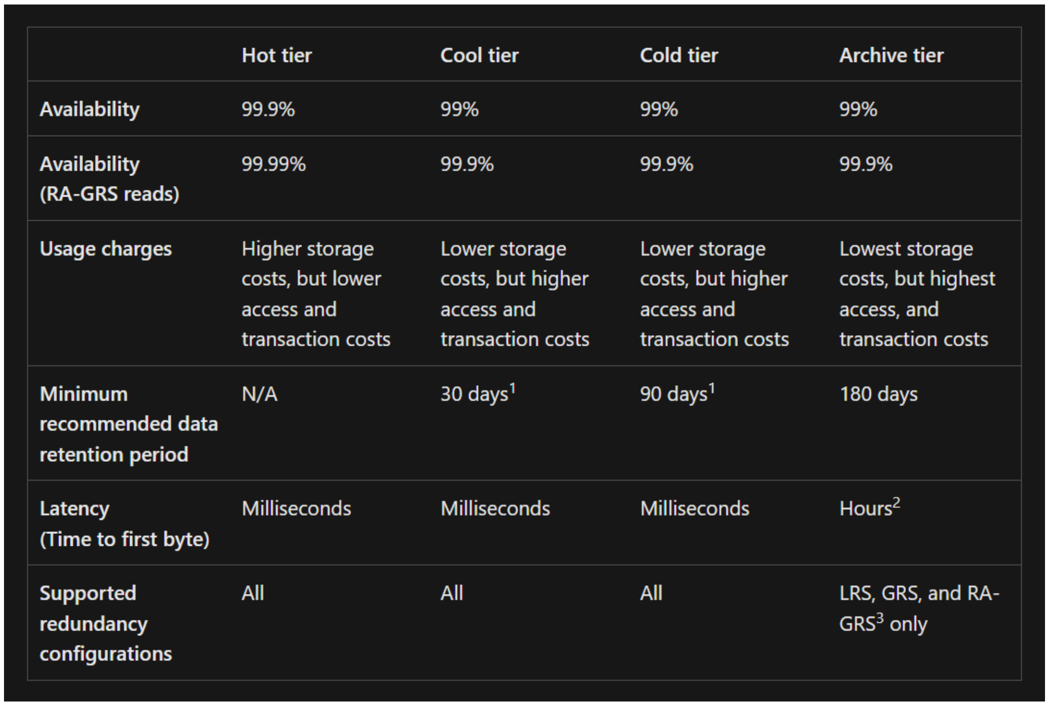
Note: - In an account that has soft delete enabled, a blob is considered deleted after it is deleted and retention period expires. Until that period expires, the blob is only *soft-deleted* and is not subject to the early deletion penalty.

Data in the archive tier can take up to 15 hours to rehydrate, depending on the priority you specify for the rehydration operation.

Storage costs for metadata of archived blobs will be charged on cool tier rates. Snapshots aren't supported for archived blobs.
The following operations are supported for blobs in the archive tier:

- Copy Blob
- Delete Blob
- Undelete Blob
- Find Blobs by Tags
- Get Blob Metadata
- Get Blob Properties
- Get Blob Tags
- List Blobs
- Set Blob Tags
- Set Blob Tier

Only storage accounts that are configured for LRS, GRS, or RA-GRS support moving blobs to the archive tier. The archive tier isn't supported for ZRS, GZRS, or RA-GZRS accounts.



	Hot tier	Cool tier	Cold tier	Archive tier
Availability	99.9%	99%	99%	99%
Availability (RA-GRS reads)	99.99%	99.9%	99.9%	99.9%
Usage charges	Higher storage costs, but lower access and transaction costs	Lower storage costs, but higher access and transaction costs	Lower storage costs, but higher access and transaction costs	Lowest storage costs, but highest access, and transaction costs
Minimum recommended data retention period	N/A	30 days[1]	90 days[1]	180 days
Latency (Time to first byte)	Milliseconds	Milliseconds	Milliseconds	Hours[2]
Supported redundancy configurations	All	All	All	LRS, GRS, and RA-GRS[3] only

Blob rehydration from the archive tier

While a blob is in the archive access tier, that blob is offline, and can't be read or modified.

To read or modify data in an archived blob, you must first rehydrate the blob to an online tier, either the hot or cool tier.

There are two options for rehydrating a blob that is stored in the archive tier:

- **Copy an archived blob to an online tier**
  You can rehydrate an archived blob by copying it to a new blob in the hot or cool tier with the Copy Blob operation.

- **Change an archived blob's access tier to an online tier**
  You can rehydrate an archived blob to the hot or cool tier by changing its tier using the Set Blob Tier operation.

Rehydrating a blob from the archive tier can *take several hours to complete*.

Microsoft recommends archiving larger blobs for optimal performance when rehydrating.
Rehydrating many small blobs might require extra time due to the processing overhead on each blob.
A *maximum of 10 GiB per storage account* may be rehydrated per hour with priority retrieval.

Rehydration priority

When you rehydrate a blob, you can set the priority for the rehydration operation via the optional x-ms-rehydrate-priority header on a Set Blob Tier or Copy Blob operation. Rehydration priority options include:

Standard priority: The rehydration request is processed in the order it was received and might take up to 15 hours to complete for objects under 10 GB in size.(default rehydration)

High priority: The rehydration request is prioritized over standard priority requests and might complete in less than one hour for objects under 10 GB in size.

Standard priority is the default rehydration option. A high-priority rehydration is faster, but also costs more than a standard-priority rehydration.

APPLICATION - DEVELOPMENT

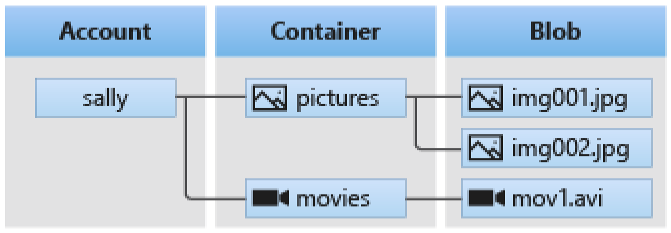
Overview of the Azure Storage client libraries

One set of libraries builds on the Azure Storage REST API, and is designed to handle data access operations for blobs, queues, and files.

Another set of libraries builds on top of the Azure Storage resource provider REST API, and is designed to handle resource management operations. These libraries are sometimes referred to as the management plane.
You can use the management plane libraries to create, update, manage, and delete resources such as storage accounts, private endpoints, and account access keys.

Understand how apps interact with Blob Storage data resources

As you build applications to work with data resources in Azure Blob Storage, your code primarily interacts with three resource types: storage accounts, containers, and blobs.



A storage account provides a unique namespace in Azure for your data. Every object that you store in Azure Storage has an address that includes your unique account name.
The combination of the account name and the Blob Storage endpoint forms the base address for the objects in your storage account.

https://sampleaccount.blob.core.windows.net

A container organizes a set of blobs, similar to a directory in a file system. A storage account can include an unlimited number of containers, and a container can store an unlimited number of blobs.
https://sampleaccount.blob.core.windows.net/sample-container

Blobs
Azure Storage supports three types of blobs:

- Block blobs store text and binary data. Block blobs are made up of blocks of data that can be managed individually. Block blobs can store up to about 190.7 TiB.

- Append blobs are made up of blocks like block blobs, but are optimized for append operations. Append blobs are ideal for scenarios such as logging data from virtual machines.
- Page blobs store random access files up to 8 TiB in size.

Create and manage client objects that interact with data resources

1. Authorize a client object
2. Create a client object
3. Create a BlobServiceClient object
4. Create a BlobContainerClient object
5. Create a BlobClient object
6. Manage client objects
7. Client immutability and thread safety

A Blob Storage endpoint forms the base address for all objects within a storage account. When you create a storage account, you specify which type of endpoint you want to use. Blob Storage supports two types of endpoints:

- A standard endpoint includes the unique storage account name along with a fixed domain name. The format of a standard endpoint is https://<storage-account>.blob.core.windows.net.

- An Azure DNS zone endpoint (preview) dynamically selects an Azure DNS zone and assigns it to the storage account when it's created.
  The format of an Azure DNS Zone endpoint is https://<storage-account>.z[00-99].blob.storage.azure.net.

When your application creates a service client object that connects to Blob Storage data resources, you pass a URI referencing the endpoint to the service client constructor. You can construct the URI string manually, or you can query for the service endpoint at runtime using the Azure Storage management library.

Install packages
Register the Storage resource provider with a subscription
Query for the Blob Storage endpoint
Create a client object using the endpoint

Change feed support in Azure Blob Storage

The purpose of the change feed is to provide transaction logs of all the changes that occur to the blobs and the blob metadata in your storage account.

The change feed provides **ordered**, **guaranteed**, **durable**, **immutable**, **read-only** logs of these changes.

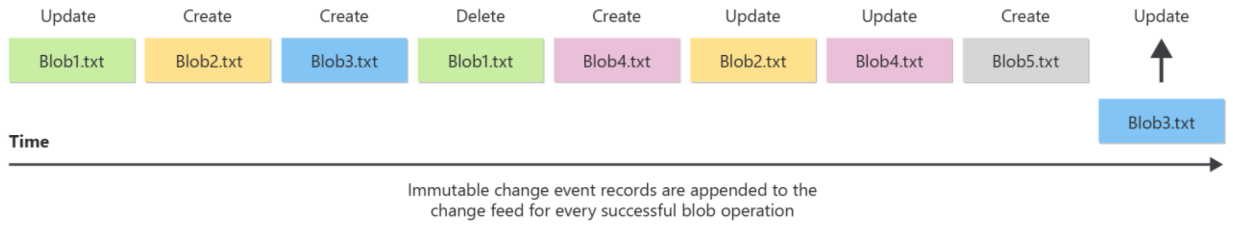
Client applications can read these logs at any time, either in streaming or in batch mode. Each change generates exactly one transaction log entry, so you won't have to manage multiple log entries for the same change.
The change feed enables you to build efficient and scalable solutions that process change events that occur in your Blob Storage account at a low cost.

How the change feed works

Change events are appended to the change feed as records in the Apache Avro format specification: a compact, fast, binary format that provides rich data structures with inline schema.

This format is widely used in the Hadoop ecosystem, Stream Analytics, and Azure Data Factory.

The following diagram shows how records are added to the change feed:



Change feed support is well-suited for scenarios that process data based on objects that have changed. For example, applications can:

- Update a secondary index, synchronize with a cache, search-engine, or any other content-management scenarios.
- Extract business analytics insights and metrics, based on changes that occur to your objects, either in a streaming manner or batched mode.
- Store, audit, and analyze changes to your objects, over any period of time, for security, compliance or intelligence for enterprise data management.
- Build solutions to backup, mirror, or replicate object state in your account for disaster management or compliance.
- Build connected application pipelines that react to change events or schedule executions based on created or changed object.

Enable and disable the change feed

You must enable the change feed on your storage account to begin capturing and recording changes. Disable the change feed to stop capturing changes.

Consume the change feed

The change feed produces several metadata and log files. These files are located in the **$blobchangefeed** container of the storage account. The **$blobchangefeed** container can be viewed either via the Azure portal or via Azure Storage Explorer.

Change feed segments
The change feed is a log of changes that are organized into **hourly** *segments* but appended to and updated every few minutes. These segments are created only when there are blob change events that occur in that hour. This enables your client application to consume changes that occur within specific ranges of time without having to search through the entire log.

Change event records
The change feed files contain a series of change event records. Each change event record corresponds to one change to an individual blob

Event record schemas
The BlobPropertiesUpdated and BlobSnapshotCreated events are currently exclusive to change feed and not yet supported for Blob Storage Events.

References:-https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blob-change-feed?tabs=azure-portal

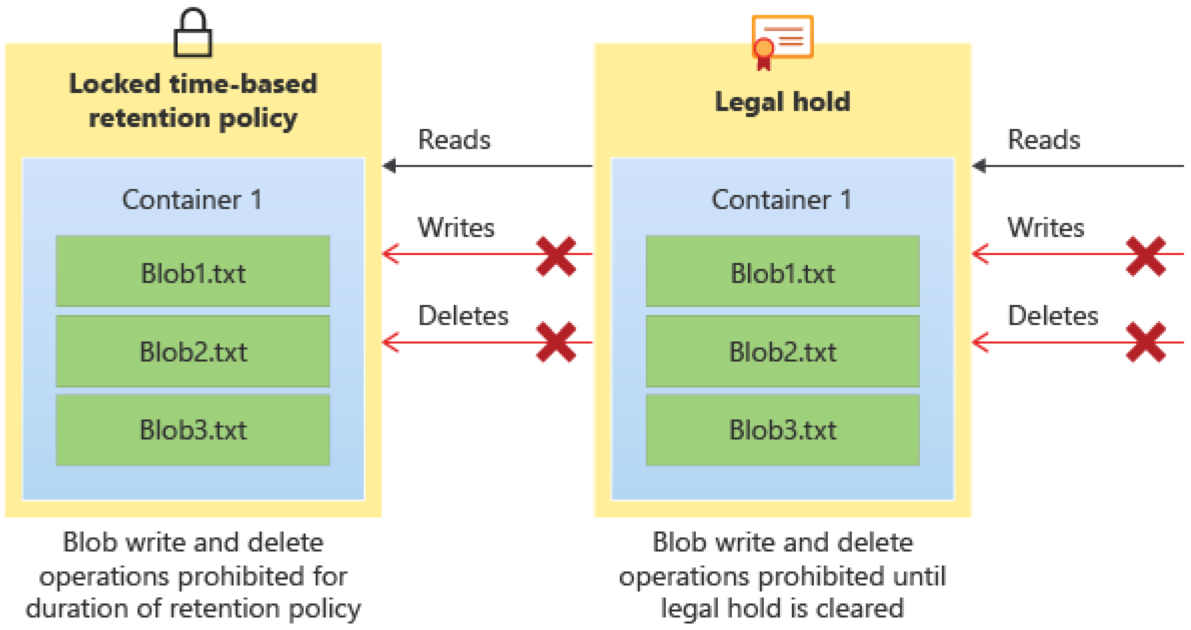
References: - https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blob-change-feed-how-to

Immutable storage for Azure Blob Storage enables users to store business-critical data in a WORM (Write Once, Read Many) state. While in a WORM state, data can't be modified or deleted for a user-specified interval. By configuring immutability policies for blob data, you can protect your data from overwrites and deletes.

Immutable storage for Azure Blob Storage supports two types of immutability policies:

- **Time-based retention policies**: With a time-based retention policy, users can set policies to store data for a specified interval. When a time-based retention policy is set, objects can be created and read, but not modified or deleted.
  After the retention period has expired, objects can be deleted but not overwritten.

- **Legal hold policies**: A legal hold stores immutable data until the legal hold is explicitly cleared. When a legal hold is set, objects can be created and read, but not modified or deleted.


Locked time-based retention policy
Container 1
Blob1.txt
Blob2.txt
Blob3.txt
Reads
Writes
Deletes
Blob write and delete operations prohibited for duration of retention policy
Legal hold
Container 1
Blob1.txt
Blob2.txt
Blob3.txt
Reads
Writes
Deletes
Blob write and delete operations prohibited until legal hold is cleared

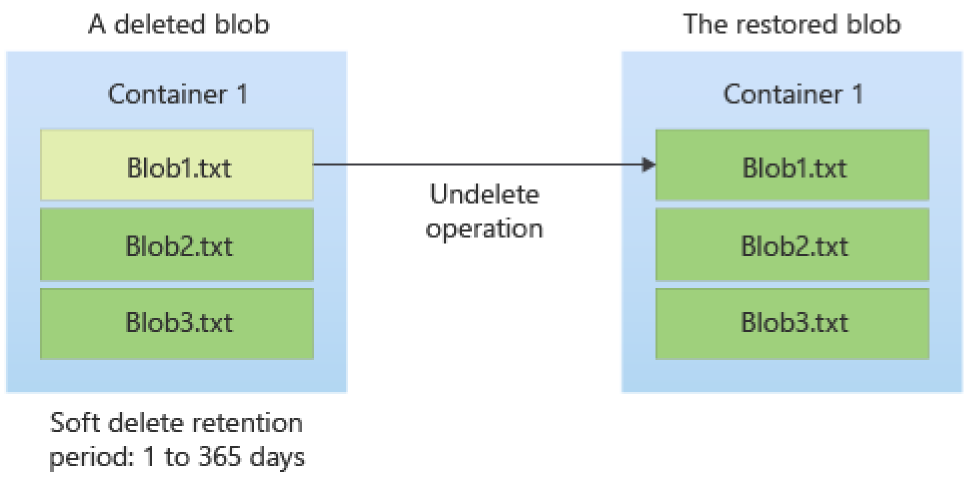
There are two features under the immutable storage umbrella: container-level WORM and version-level WORM.
Container-level WORM allows policies to be set at the container level only, while version-level WORM allows policies to be set at the account, container, or version level.

Soft delete for blobs

Blob soft delete protects an individual blob, snapshot, or version from accidental deletes or overwrites by maintaining the deleted data in the system for a specified period of time. During the retention period, you can restore a soft-deleted object to its state at the time it was deleted. After the retention period has expired, the object is permanently deleted.

How blob soft delete works

When you enable blob soft delete for a storage account, you specify a retention period for deleted objects of between 1 and 365 days. The retention period indicates how long the data remains available after it's deleted or overwritten. The clock starts on the retention period as soon as an object is deleted or overwritten.
While the retention period is active, you can restore a deleted blob, together with its snapshots, or a deleted version by calling the Undelete Blob operation. The following diagram shows how a deleted object can be restored when blob soft delete is enabled:



You can change the soft delete retention period at any time. An updated retention period applies only to data that was deleted after the retention period was changed. Any data that was deleted before the retention period was changed is subject to the retention period that was in effect when it was deleted.
Attempting to delete a soft-deleted object doesn't affect its expiry time.
If you disable blob soft delete, you can continue to access and recover soft-deleted objects in your storage account until the soft delete retention period has elapsed.

Manage blob properties and metadata with .NET

Set up your environment
Install packages

```
using Azure.Identity;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
using Azure.Storage.Blobs.Specialized;
```

Create a client object

```
public BlobServiceClient GetBlobServiceClient(string accountName)
{
   BlobServiceClient client = new(
      new Uri($"https://{accountName}.blob.core.windows.net"),
      new DefaultAzureCredential());

   return client;
}
```

About properties and metadata

- **System properties**: System properties exist on each Blob storage resource. Some of them can be read or set, while others are read-only. Under the covers, some system properties correspond to certain standard HTTP headers. The Azure Storage client library for .NET maintains these properties for you.

- **User-defined metadata**: User-defined metadata consists of one or more name-value pairs that you specify for a Blob storage resource. You can use metadata to store additional values with the resource. Metadata values are for your own purposes only, and don't affect how the resource behaves.

Metadata name/value pairs are valid HTTP headers and should adhere to all restrictions governing HTTP headers. For more information about metadata naming requirements, see Metadata names.

Set and retrieve properties

To set properties on a blob, call SetHttpHeaders or SetHttpHeadersAsync. Any properties not explicitly set are cleared. The following code example first gets the existing properties on the blob, then uses them to populate the headers that aren't being updated.

Set and retrieve metadata

You can specify metadata as one or more name-value pairs on a blob or container resource. To set metadata, add name-value pairs to the Metadata collection on the resource. Then, call one of the following methods to write the values:

- SetMetadata
- SetMetadataAsync

The following code example sets metadata on a blob. One value is set using the collection's Add method. The other value is set using implicit key/value syntax.

```
public static async Task AddBlobMetadataAsync(BlobClient blob)
{
    Console.WriteLine("Adding blob metadata...");

    try
    {
        IDictionary<string, string> metadata =
           new Dictionary<string, string>();

        // Add metadata to the dictionary by calling the Add method
        metadata.Add("docType", "textDocuments");

        // Add metadata to the dictionary by using key/value syntax
        metadata["category"] = "guidance";

        // Set the blob's metadata.
        await blob.SetMetadataAsync(metadata);
    }
    catch (RequestFailedException e)
    {
        Console.WriteLine($"HTTP error code {e.Status}: {e.ErrorCode}");
        Console.WriteLine(e.Message);
        Console.ReadLine();
    }
}
```

To retrieve metadata, call the GetProperties or GetPropertiesAsync method on your blob or container to populate the Metadata collection, then read the values, as shown in the example below. The GetProperties method retrieves blob properties and metadata by calling both the Get Blob Properties operation and the Get Blob Metadata operation.

Authorize access to blobs using Microsoft Entra ID

Azure Storage supports using Microsoft Entra ID to authorize requests to blob data. With Microsoft Entra ID, you can use Azure role-based access control (Azure RBAC) to grant permissions to a security principal, which may be a user, group, or application service principal.

The security principal is authenticated by Microsoft Entra ID to return an OAuth 2.0 token.

Use Microsoft Entra ID to authorize access in application code
To authorize access to Azure Storage with Microsoft Entra ID, you can use one of the following client libraries to acquire an OAuth 2.0 token:

- The Azure Identity client library is recommended for most development scenarios.
- The Microsoft Authentication Library (MSAL) may be suitable for certain advanced scenarios

Storage Blob Data Owner: Use to set ownership and manage POSIX access control for Azure Data Lake Storage. For more information, see Access control in Azure Data Lake Storage.

Storage Blob Data Contributor: Use to grant read/write/delete permissions to Blob storage resources.

Storage Blob Data Reader: Use to grant read-only permissions to Blob storage resources.

Storage Blob Delegator: Get a user delegation key to use to create a shared access signature that is signed with Microsoft Entra credentials for a container or blob.

Revoke a user delegation SAS

If you believe that a SAS has been compromised, you should revoke it. You can revoke a user delegation SAS either by revoking the user delegation key, or by changing or removing RBAC role assignments and POSIX ACLs for the security principal that's used to create the SAS.

- **Revoke the user delegation key**
  You can revoke the user delegation key by calling the Revoke User Delegation Keys operation. When you revoke the user delegation key, any shared access signatures that rely on that key become invalid. You can then call the Get User Delegation Key operation again and use the key to create new shared access signatures. This is the quickest way to revoke a user delegation SAS.

- **Change or remove role assignments or ACLs**
  You can change or remove the RBAC role assignment and POSIX ACLs for the security principal that's used to create the SAS. When a client uses the SAS to access a resource, Azure Storage verifies that the security principal whose credentials were used to secure the SAS has the required permissions to the resource.

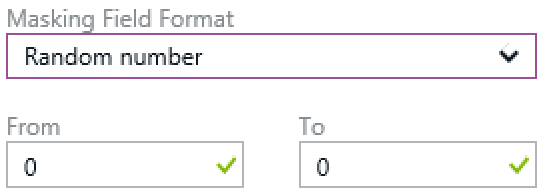
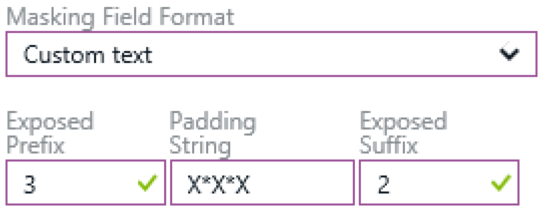
Dynamic data masking

Azure SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics support dynamic data masking (DDM). Dynamic data masking limits sensitive data exposure by masking it to nonprivileged users.

Dynamic data masking helps prevent unauthorized access to sensitive data by enabling customers to designate how much of the sensitive data to reveal with minimal effect on the application layer. It's a policy-based security feature that hides the sensitive data in the result set of a query over designated database fields, while the data in the database isn't changed.

For example, a service representative at a call center might identify a caller by confirming several characters of their email address, but the complete email address shouldn't be revealed to the service representative. A masking rule can be defined that masks all the email address in the result set of any query. As another example, an appropriate data mask can be defined to protect personal data, so that a developer can query production environments for troubleshooting purposes without violating compliance regulations.

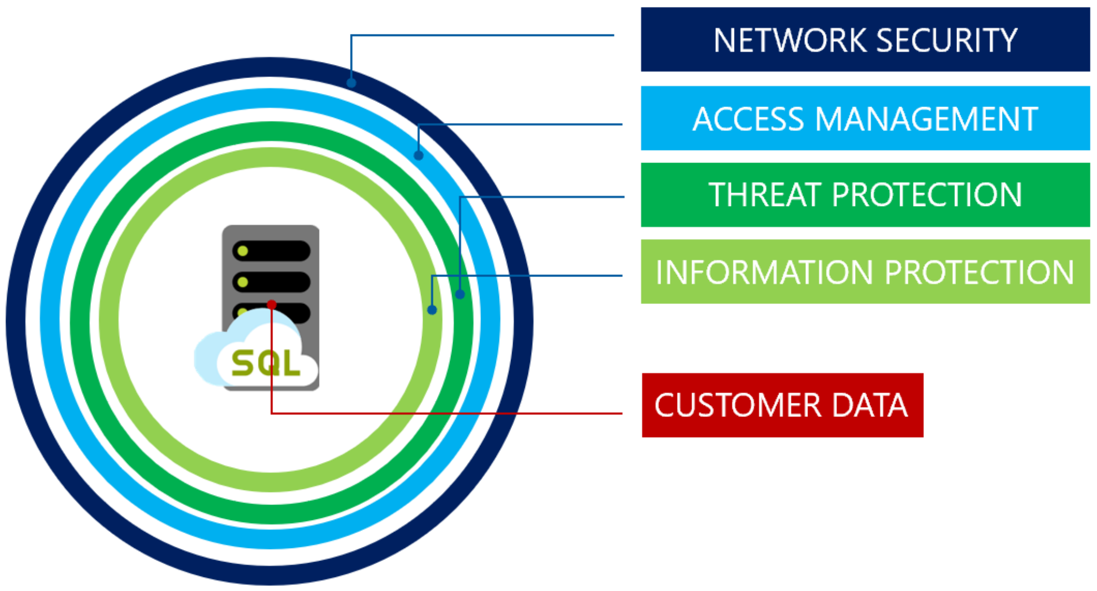
Masking function	Masking logic
Default	**Full masking according to the data types of the designated fields** * Use XXXX (or fewer) if the size of the field is fewer than 4 characters for string data types (**nchar**, **ntext**, **nvarchar**).

	* Use a zero value for numeric data types (**bigint**, **bit**, **decimal**, **int**, **money**, **numeric**, **smallint**, **smallmoney**, **tinyint**, **float**, **real**). * Use 1900-01-01 for date/time data types (**date**, **datetime2**, **datetime**, **datetimeoffset**, **smalldatetime**, **time**). * For **sql_variant**, the default value of the current type is used. * For XML, the document <masked /> is used. * Use an empty value for special data types (**timestamp**, **table**, **HierarchyID**, **uniqueidentifier**, **binary**, **image**, **varbinary**, and spatial types).
Credit card	**Masking method, which exposes the last four digits of the designated fields** and adds a constant string as a prefix in the form of a credit card. XXXX-XXXX-XXXX-1234
Email	**Masking method, which exposes the first letter and replaces the domain with XXX.com** using a constant string prefix in the form of an email address. aXX@XXXX.com
Random number	**Masking method, which generates a random number** according to the selected boundaries and actual data types. If the designated boundaries are equal, then the masking function is a constant number.  Masking Field Format: Random number From: 0 To: 0
Custom text	**Masking method, which exposes the first and last characters** and adds a custom padding string in the middle. If the original string is shorter than the exposed prefix and suffix, only the padding string is used. prefix[padding]suffix  Masking Field Format: Custom text Exposed Prefix: 3 Padding String: X*X*X Exposed Suffix: 2

Recommended fields to mask

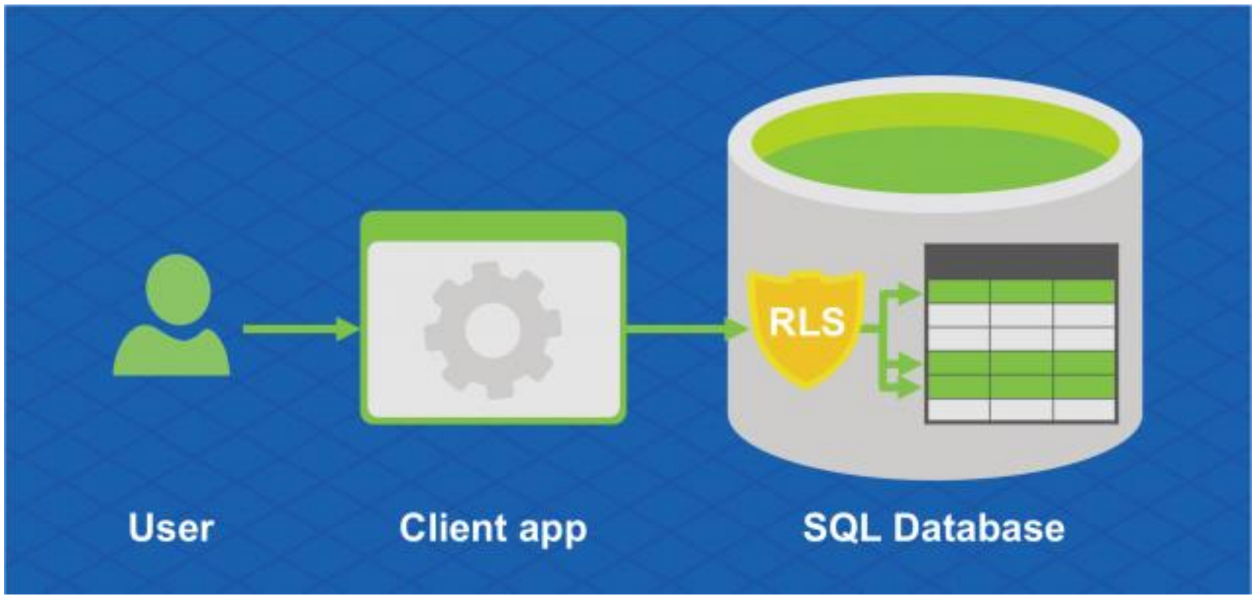
The DDM recommendations engine flags certain fields from your database as potentially sensitive fields, which might be good candidates for masking. In the **Dynamic Data Masking** pane in the portal, you see the recommended columns for your database. Select **Add Mask** for one or more columns, then select the appropriate masking function and select **Save**, to apply mask for these fields.

An overview of Azure SQL Database and SQL Managed Instance security capabilities



Row-level security

Row-Level Security enables customers to control access to rows in a database table based on the characteristics of the user executing a query (for example, group membership or execution context). Row-Level Security can also be used to implement custom Label-based security concepts. For more information, see Row-Level security.



Threat protection

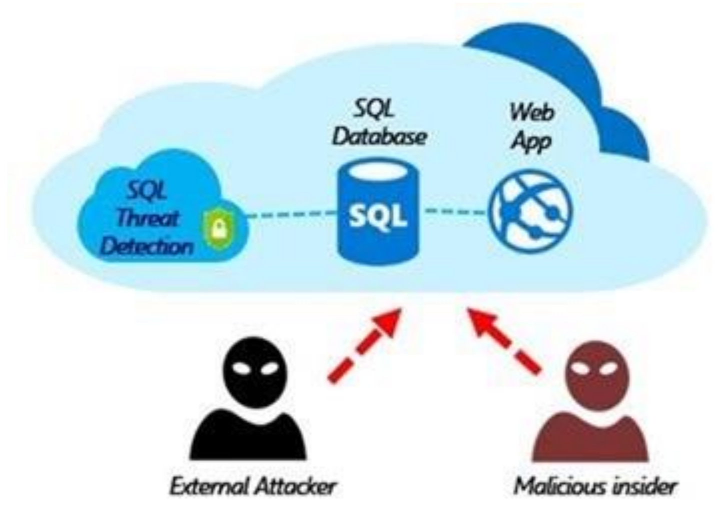
SQL Database and SQL Managed Instance secure customer data by providing auditing and threat detection capabilities.

SQL auditing in Azure Monitor logs and Event Hubs

SQL Database and SQL Managed Instance auditing tracks database activities and helps maintain compliance with security standards by recording database events to an audit log in a customer-owned Azure storage account. Auditing allows users to monitor ongoing database activities, as well as analyze and investigate historical activity to identify potential threats or suspected abuse and security violations. For more information, see Get started with SQL Database Auditing.

Advanced Threat Protection

Advanced Threat Protection is analyzing your logs to detect unusual behavior and potentially harmful attempts to access or exploit databases. Alerts are created for suspicious activities such as SQL injection, potential data infiltration, and brute force attacks or for anomalies in access patterns to catch privilege escalations and breached credentials use. Alerts are viewed from the Microsoft Defender for Cloud, where the details of the suspicious activities are provided and recommendations for further investigation given along with actions to mitigate the threat. Advanced Threat Protection can be enabled per server for an additional fee. For more information, see Get started with SQL Database Advanced Threat Protection.



Information protection and encryption

Transport Layer Security (Encryption-in-transit)

SQL Database, SQL Managed Instance, and Azure Synapse Analytics secure customer data by encrypting data in motion with Transport Layer Security (TLS).

SQL Database, SQL Managed Instance, and Azure Synapse Analytics enforce encryption (SSL/TLS) at all times for all connections. This ensures all data is encrypted "in transit" between the client and server irrespective of the setting of **Encrypt** or **TrustServerCertificate** in the connection string.

As a best practice, recommend that in the connection string used by the application, you specify an encrypted connection and ***not*** trust the server certificate. This forces your application to verify the server certificate and thus prevents your application from being vulnerable to man in the middle type attacks.

For example when using the ADO.NET driver this is accomplished via **Encrypt=True** and **TrustServerCertificate=False**. If you obtain your connection string from the Azure portal, it will have the correct settings.

Important

Note that some non-Microsoft drivers may not use TLS by default or rely on an older version of TLS (<1.2) in order to function. In this case the server still allows you to connect to your database. However, we recommend that you evaluate the security risks of allowing such drivers and application to connect to SQL Database, especially if you store sensitive data.

For further information about TLS and connectivity, see **TLS considerations**

Transparent Data Encryption (Encryption-at-rest)

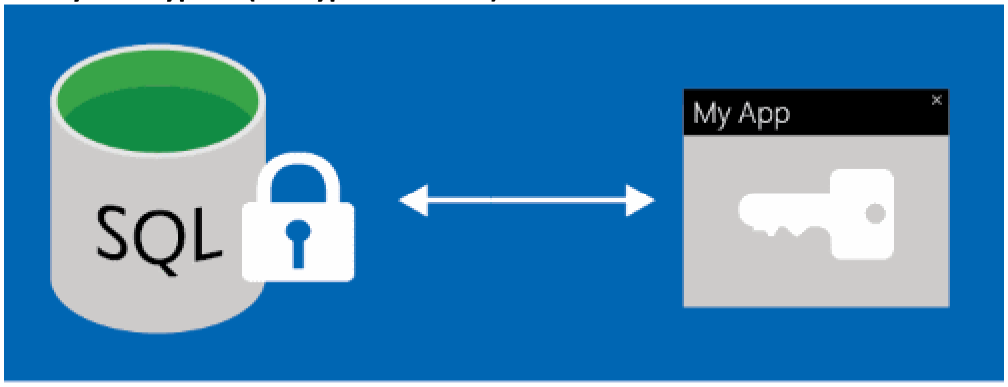
Transparent data encryption (TDE) for SQL Database, SQL Managed Instance, and Azure Synapse Analytics adds a layer of security to help protect data at rest from unauthorized or offline access to raw files or backups. Common scenarios include data center theft or unsecured disposal of hardware or media such as disk drives and backup tapes. TDE encrypts the entire database using an AES encryption algorithm, which doesn't require application developers to make any changes to existing applications.

In Azure, all newly created databases are encrypted by default and the database encryption key is protected by a built-in server certificate. Certificate maintenance and rotation are managed by the service and require no input from the user. Customers who prefer to take control of the encryption keys can manage the keys in Azure Key Vault.

Key management with Azure Key Vault

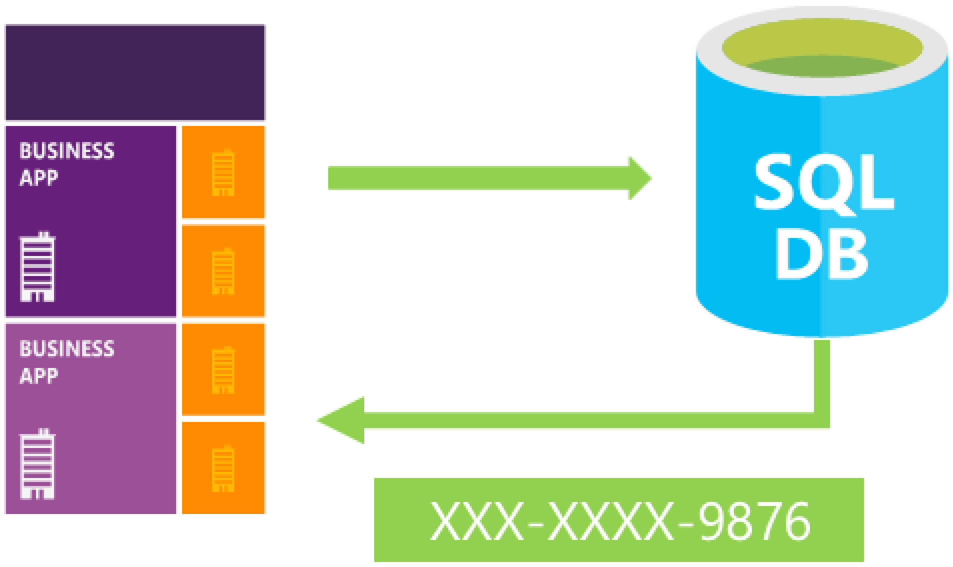
Bring Your Own Key (BYOK) support for Transparent Data Encryption (TDE) allows customers to take ownership of key management and rotation using Azure Key Vault, Azure's cloud-based external key management system. If the database's access to the key vault is revoked, a database cannot be decrypted and read into memory. Azure Key Vault provides a central key management platform, leverages tightly monitored hardware security modules (HSMs), and enables separation of duties between management of keys and data to help meet security compliance requirements.

Always Encrypted (Encryption-in-use)



Always Encrypted is a feature designed to protect sensitive data stored in specific database columns from access (for example, credit card numbers, national/regional identification numbers, or data on a *need to know* basis). This includes database administrators or other privileged users who are authorized to access the database to perform management tasks, but have no business need to access the particular data in the encrypted columns. The data is always encrypted, which means the encrypted data is decrypted only for processing by client applications with access to the encryption key. The encryption key is never exposed to SQL Database or SQL Managed Instance and can be stored either in the Windows Certificate Store or in Azure Key Vault.

Dynamic data masking



Dynamic data masking limits sensitive data exposure by masking it to non-privileged users. Dynamic data masking automatically discovers potentially sensitive data in Azure SQL Database and SQL Managed Instance and provides actionable recommendations to mask these fields, with minimal impact to the application layer.