# Azure Blob Storage

## Explore Azure Blob storage

Azure Blob storage is Microsoft's object storage solution for the cloud.

Blob storage is optimized for storing massive amounts of unstructured data.

Unstructured data is data that does not 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.

### Access

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.

### Types of storage accounts

Azure Storage offers two performance levels of storage accounts, standard and premium. Each performance level supports different features and has its own pricing model.

* **Standard:** This is the standard general-purpose v2 account and is recommended for most scenarios using Azure Storage.
* **Premium:** Premium accounts offer higher performance by using solid-state drives. If you create a premium account you can choose between three account types, block blobs, page blobs, or file shares.

### Access tiers for block blob data

1. The Hot access tier:

Is optimized for frequent access of objects in the storage account. The Hot tier has the highest storage costs, but the lowest access costs. New storage accounts are created in the hot tier by default.

1. The Cool access tier:

Which is optimized for storing large amounts of data that is infrequently accessed and stored for at least 30 days. The Cool tier has lower storage costs and higher access costs compared to the Hot tier.

1. The Archive tier:

Which is available only for individual block blobs. The archive tier is optimized for data that can tolerate several hours of retrieval latency and will remain in the Archive tier for at least 180 days. The archive tier is the most cost-effective option for storing data, but accessing that data is more expensive than accessing data in the hot or cool tiers.

**If there is a change in the usage pattern of your data, you can switch between these access tiers at any time.**

## Blob storage resource types

### Storage accounts

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.

For example, if your storage account is named “*mystorageaccount”*, then the default endpoint for Blob storage is: [*http://mystorageaccount.blob.core.windows.net*](http://mystorageaccount.blob.core.windows.net)*.*

### Containers

A container organizes a set of blobs, like 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. The container name must be lowercase.

### Blobs

Azure Storage supports three types of blobs:

* **Block blobs** store text and binary data, up to about 190.7 TB. Block blobs are made up of blocks of data that can be managed individually.
* **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 TB in size. Page blobs store virtual hard drive (VHD) files and serve as disks for Azure virtual machines.

## Azure Storage security features

Azure Storage provides a comprehensive set of security capabilities that together enable developers to build secure applications:

* All data (including metadata) written to Azure Storage is automatically encrypted using Storage Service Encryption (SSE).
* Azure Active Directory (Azure AD) and Role-Based Access Control (RBAC) are supported for Azure Storage for both resource management operations and data operations, as follows:
  + You can assign RBAC roles scoped to the storage account to security principals and use Azure AD to authorize resource management operations such as key management.
  + Azure AD integration is supported for blob and queue data operations. You can assign RBAC roles scoped to a subscription, resource group, storage account, or an individual container or queue to a security principal or a managed identity for Azure resources.
* Data can be secured in transit between an application and Azure by using Client-Side Encryption, HTTPS, or SMB 3.0.
* OS and data disks used by Azure virtual machines can be encrypted using Azure Disk Encryption.
* Delegated access to the data objects in Azure Storage can be granted using a shared access signature.

### Azure Storage encryption for data at rest

Azure Storage automatically encrypts your data when persisting it to the cloud. Encryption protects your data and help you meet **your organizational security and compliance commitments**. Data in Azure Storage is encrypted and decrypted transparently using **256-bit AES encryption**, one of the strongest block ciphers available, and is **FIPS 140-2 compliant**. Azure Storage encryption is similar to **BitLocker encryption on Windows.**

**Azure Storage encryption is enabled for all new and existing storage accounts and cannot be disabled.** Because your data is secured by default, you **don't need to modify your code** or applications to take advantage of Azure Storage encryption.

Storage accounts are encrypted **regardless of their performance tier** (standard or premium) or deployment model (Azure Resource Manager or classic). All Azure Storage **redundancy options** support encryption, and **all copies of a storage account are encrypted**.

**All Azure Storage resources** are encrypted, including **blobs, disks, files, queues**, and **tables**. **All object metadata** is also encrypted.

**Encryption does not affect Azure Storage performance.**

**There is no additional cost for Azure Storage encryption.**

### Encryption key management

* You can rely on ***Microsoft-managed keys*** for the encryption of your storage account, or you can manage encryption with your own keys.

If you choose to manage encryption with your own keys, you have two options:

* You can specify a ***customer-managed*** ***key*** to use for encrypting and decrypting all data in the storage account. A customer-managed key is used to encrypt all data in all services in your storage account.
* You can specify a ***customer-provided key*** on Blob storage operations. A client making a read or write request against Blob storage can include an encryption key on the request for granular control over how blob data is encrypted and decrypted.

The following table compares key management options for Azure Storage encryption.

|  | **Microsoft-managed keys** | **Customer-managed keys** | **Customer-provided keys** |
| --- | --- | --- | --- |
| Encryption / decryption operations | Azure | Azure | Azure |
| Azure Storage services supported | All | Blob storage, Azure Files | Blob storage |
| Key storage | Microsoft key store | Azure Key Vault | Azure Key Vault or any other key store |
| Key rotation responsibility | Microsoft | Customer | Customer |
| Key usage | Microsoft | Azure portal, Storage Resource Provider REST API, Azure Storage management libraries, PowerShell, CLI | Azure Storage REST API (Blob storage), Azure Storage client libraries |
| Key access | Microsoft only | Microsoft, Customer | Customer only |

## Azure Storage redundancy options

Azure Storage always stores multiple copies of your data so that it is protected from planned and unplanned events, including transient hardware failures, network or power outages, and massive natural disasters. Redundancy ensures that your storage account meets its availability and durability targets even in the face of failures.

### Redundancy in the primary region

Data in an Azure Storage account is always replicated three times in the primary region. Azure Storage offers two options for how your data is replicated in the primary region.

* Locally redundant storage (LRS):

Copies your data synchronously three times within a single physical location in the primary region. LRS is the least expensive replication option, but is not recommended for applications requiring high availability or durability.

* Zone-redundant storage (ZRS):

Copies your data synchronously across three Azure availability zones in the primary region. For applications requiring high availability, Microsoft recommends using ZRS in the primary region, and also replicating to a secondary region.

### Redundancy in a secondary region

For applications requiring high durability, you can choose to additionally copy the data in your storage account to a secondary region that is hundreds of miles away from the primary region. If your storage account is copied to a secondary region, then your data is durable even in the case of a complete regional outage or a disaster in which the primary region isn't recoverable.

When you create a storage account, you select the primary region for the account. The paired secondary region is determined based on the primary region, and can't be changed.

Azure Storage offers two options for copying your data to a secondary region:

* Geo-redundant storage (GRS)

Copies your data synchronously three times within a single physical location in the primary region using LRS. It then copies your data asynchronously to a single physical location in the secondary region. Within the secondary region, your data is copied synchronously three times using LRS.

* Geo-zone-redundant storage (GZRS)

Copies your data synchronously across three Azure availability zones in the primary region using ZRS. It then copies your data asynchronously to a single physical location in the secondary region. Within the secondary region, your data is copied synchronously three times using LRS.

## Create a block blob storage account

Azure portal -> All services -> the Storage category -> Storage accounts

-> select + Create

-> {

“Subscription” = subscription in which to create the storage account

“Resource group field” = Create new and enter the new resource group

“Storage account name” -> Enter a name for the account.

“Location” -> Select a location for the storage account, or use the default location

“Performance” -> Select Premium.

“Premium account type” -> Select Block blobs.

“Replication” -> Leave the default setting of Locally-redundant storage (LRS).

}

-> Select Review + create to review the storage account settings.

-> Select Create.

# Azure Blob storage lifecycle

Data sets have unique lifecycles. Early in the lifecycle, people access some data often. But the need for access drops drastically as the data ages. Some data stays idle in the cloud and is rarely accessed once stored.

Some data expires days or months after creation, while other data sets are actively read and modified throughout their lifetimes.

## Access tiers

Azure storage offers different access tiers, allowing you to store blob object data in the most cost-effective manner. Available access tiers include:

1. Hot - Optimized for storing data that is accessed frequently.
2. Cool - Optimized for storing data that is infrequently accessed and stored for at least 30 days.
3. Archive - Optimized for storing data that is rarely accessed and stored for at least 180 days with flexible latency requirements, on the order of hours.

The following considerations apply to the different access tiers:

* The access tier can be set on a blob during or after upload.
* **Only the hot and cool access tiers can be set at the account level. The archive access tier can only be set at the blob level.**
* Data in the cool access tier has slightly lower availability, but still has high durability, retrieval latency, and throughput characteristics similar to hot data.
* Data in the archive access tier is stored offline. The archive tier offers the lowest storage costs but also the highest access costs and latency.
* The hot and cool tiers support all redundancy options. The archive tier supports only LRS, GRS, and RA-GRS.
* Data storage limits are set at the account level and not per access tier. You can choose to use all of your limit in one tier or across all three tiers.

## Manage the data lifecycle

Azure Blob storage lifecycle management offers a rich, rule-based policy for General Purpose v2 and Blob storage accounts. Use the policy to transition your data to the appropriate access tiers or expire at the end of the data's lifecycle. The lifecycle management policy lets you:

* Transition blobs to a cooler storage tier (hot to cool, hot to archive, or cool to archive) to optimize for performance and cost
* Delete blobs at the end of their lifecycles
* Define rules to be run once per day at the storage account level
* Apply rules to containers or a subset of blobs (using prefixes as filters)

## Blob storage lifecycle policies

A lifecycle management policy is a collection of rules in a JSON document. Each rule definition within a policy includes a filter set and an action set. The filter set limits rule actions to a certain set of objects within a container or objects names. The action set applies the tier or delete actions to the filtered set of objects.



### Policy

A policy is a collection of rules:

| **Parameter name** | **Parameter type** | **Notes** |
| --- | --- | --- |
| rules | An array of rule objects | At least one rule is required in a policy. You can define up to 100 rules in a policy. |

### Rules

Each rule within the policy has several parameters:

| **Parameter name** | **Parameter type** | **Notes** | **Required** |
| --- | --- | --- | --- |
| name | String | A rule name can include up to 256 alphanumeric characters. Rule name is case-sensitive. It must be unique within a policy. | True |
| enabled | Boolean | An optional boolean to allow a rule to be temporary disabled. Default value is true if it's not set. | False |
| type | An enum value | The current valid type is Lifecycle. | True |
| definition | An object that defines the lifecycle rule | Each definition is made up of a **filter** set and an **action** set. | True |

**Each rule definition includes** a **“filter”** set and an **“action”** set. The filter set limits rule actions to a certain set of objects within a container or objects names. The action set applies the tier or delete actions to the filtered set of objects.

### Rule filters

Filters limit rule actions to a subset of blobs within the storage account. If more than one filter is defined, a logical AND runs on all filters. Filters include:

| **Filter name** | **Filter type** | **Is Required** |
| --- | --- | --- |
| blobTypes | An array of predefined enum values. | Yes |
| prefixMatch | An array of strings for prefixes to be match. Each rule can define up to 10 prefixes. A prefix string must start with a container name. | No |
| blobIndexMatch | An array of dictionary values consisting of blob index tag key and value conditions to be matched. Each rule can define up to 10 blob index tag condition. | No |

### Rule actions

Actions are applied to the filtered blobs when the run condition is met. Lifecycle management supports tiering and deletion of blobs and deletion of blob snapshots.

Define at least one action for each rule on blobs or blob snapshots.

| **Action** | **Base Blob** | **Snapshot** | **Version** |
| --- | --- | --- | --- |
| tierToCool | Supported for blockBlob | Supported | Supported |
| enableAutoTierToHotFromCool | Supported for blockBlob | Not supported | Not supported |
| tierToArchive | Supported for blockBlob | Supported | Supported |
| delete | Supported for blockBlob and appendBlob | Supported | Supported |

### Run conditions

The run conditions are based on age. Base blobs use the last modified time to track age, and blob snapshots use the snapshot creation time to track age.

| **Action run condition** | **Condition value** | **Description** |
| --- | --- | --- |
| daysAfterModificationGreaterThan | Integer value indicating the age in days | The condition for base blob actions |
| daysAfterCreationGreaterThan | Integer value indicating the age in days | The condition for blob snapshot actions |

## Implement Blob storage lifecycle policies

You can add, edit, or remove a policy by using any of the following methods:

* Azure portal
* Azure PowerShell
* Azure CLI
* REST APIs

Below are the steps and some examples for the Portal.

There are two ways to add a policy through the Azure portal: Azure portal List view, and Azure portal Code view.

### Azure portal List view

Azure portal -> All resources -> Your storage account -> Under Data management

-> Lifecycle management

-> List view tab -> Add rule -> Fill Action set form fields.

-> Filter (add an optional filter -> Browse to specify a container and folder by which to filter)

-> Select Review + add to review the policy settings.

-> Add.

### Azure portal Code view

In the following example, blobs are moved to cool storage if they haven't been modified for 30 days.

1. Follow the first three steps above in the **List view** section.
2. Select the **Code view** tab. The following JSON is an example of a policy that moves a block blob whose name begins with *log* to the cool tier if it has been more than 30 days since the blob was modified.



1. Select Save.

## Rehydrate blob data from the archive tier

While a blob is in the archive access tier, it's considered to be 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](https://docs.microsoft.com/en-us/rest/api/storageservices/copy-blob) or [Copy Blob from URL](https://docs.microsoft.com/en-us/rest/api/storageservices/copy-blob-from-url) operation. Microsoft recommends this option for most scenarios.

* Change a blob's access tier to an online tier:

You can rehydrate an archived blob to hot or cool by changing its tier using the [Set Blob Tier](https://docs.microsoft.com/en-us/rest/api/storageservices/set-blob-tier) operation.

*Rehydrating a blob from the archive tier can take several hours to complete. Microsoft recommends rehydrating larger blobs for optimal performance. Rehydrating several small blobs concurrently may require additional time.*

### 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](https://docs.microsoft.com/en-us/rest/api/storageservices/set-blob-tier) or Copy Blob/Copy Blob From URL operation.

Rehydration priority options include:

* Standard priority: The rehydration request will be processed in the order it was received and may take up to 15 hours.
* High priority: The rehydration request will be prioritized over standard priority requests and may complete in under one hour for objects under 10 GB in size.

To check the rehydration priority while the rehydration operation is underway, call [Get Blob Properties](https://docs.microsoft.com/en-us/rest/api/storageservices/get-blob-properties) to return the value of the x-ms-rehydrate-priority header. The rehydration priority property returns either *Standard* or *High*.

### Copy an archived blob to an online tier

You can use either the **Copy Blob** or **Copy Blob from URL** operation to copy the blob. When you copy an archived blob to a new blob an online tier, the source blob remains unmodified in the archive tier.

You must copy the archived blob to a new blob with a different name or to a different container. You cannot overwrite the source blob by copying to the same blob.

Copying an archived blob to an online destination tier is supported within the same storage account only. You cannot copy an archived blob to a destination blob that is also in the archive tier.

The following table shows the behavior of a blob copy operation, depending on the tiers of the source and destination blob.

|  | **Hot tier source** | **Cool tier source** | **Archive tier source** |
| --- | --- | --- | --- |
| Hot tier destination | Supported | Supported | Supported within the same account. Requires blob rehydration. |
| Cool tier destination | Supported | Supported | Supported within the same account. Requires blob rehydration. |
| Archive tier destination | Supported | Supported | Unsupported |

### Change a blob's access tier to an online tier

The second option for rehydrating a blob from the archive tier to an online tier is to change the blob's tier by calling **Set Blob Tier**. With this operation, you can change the tier of the archived blob to either hot or cool.

Once a **Set Blob Tier** request is initiated, it cannot be canceled. During the rehydration operation, the blob's access tier setting continues to show as archived until the rehydration process is complete.

To learn how to rehydrate a blob by changing its tier to an online tier, see [Rehydrate a blob by changing its tier](https://docs.microsoft.com/en-us/azure/storage/blobs/archive-rehydrate-to-online-tier#rehydrate-a-blob-by-changing-its-tier).

### Caution:

Changing a blob's tier doesn't affect its last modified time. If there is a lifecycle management policy in effect for the storage account, then rehydrating a blob with **Set Blob Tier** can result in a scenario where the lifecycle policy moves the blob back to the archive tier after rehydration because the last modified time is beyond the threshold set for the policy.

# Work with Azure Blob storage

## Azure Blob storage client library

The Azure Storage client libraries for .NET offer a convenient interface for making calls to Azure Storage. The latest version of the Azure Storage client library is version 12.x. Microsoft recommends using version 12.x for new applications.

Below are the classes in the Azure.Storage.Blobs namespace and their purpose:

| **Class** | **Description** |
| --- | --- |
| BlobClient | The BlobClient allows you to manipulate Azure Storage blobs. |
| BlobClientOptions | Provides the client configuration options for connecting to Azure Blob Storage. |
| BlobContainerClient | The BlobContainerClient allows you to manipulate Azure Storage containers and their blobs. |
| BlobServiceClient | The BlobServiceClient allows you to manipulate Azure Storage service resources and blob containers. The storage account provides the top-level namespace for the Blob service. |
| BlobUriBuilder | The BlobUriBuilder class provides a convenient way to modify the contents of a Uri instance to point to different Azure Storage resources like an account, container, or blob. |

## Blob storage resources by using the .NET client library

This exercise uses the Azure Blob storage client library to show you how to perform the following actions on Azure Blob storage in a console app:

* Create a container
* Upload blobs to a container
* List the blobs in a container
* Download blobs
* Delete a container

### Nugget Package used:

**Azure.Storage.Blobs**

[Find the code here](https://github.com/AjinkyaApte88/General/tree/main/mslearn-storage-blob).

## Manage container properties and metadata by using .NET

Blob containers support system properties and user-defined metadata, in addition to the data they contain.

* **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 do not affect how the resource behaves.

### Retrieve container properties

To retrieve container properties, call one of the following methods of the **BlobContainerClient** class:

* **GetProperties**
* **GetPropertiesAsync**

### 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 an IDictionary object, and then call one of the following methods of the **BlobContainerClient** class to write the values:

1. **SetMetadata**
2. **SetMetadataAsync**

To retrieve metadata, use:

* **GetProperties**
* **GetPropertiesAsync**

## Set and retrieve properties and metadata for blob resources by using REST

Containers and blobs support custom metadata, represented as HTTP headers. Metadata headers can be set on a request that creates a new container or blob resource, or on a request that explicitly creates a property on an existing resource.

### Metadata header format

Metadata headers are name/value pairs. The format for the header is:

**x-ms-meta-name:string-value**

### Operations on metadata

Metadata on a blob or container resource can be retrieved or set directly, without returning or altering the content of the resource.

Note that metadata values can only be read or written in full; partial updates are not supported. Setting metadata on a resource overwrites any existing metadata values for that resource.

### Retrieving properties and metadata

The GET and HEAD operations both retrieve metadata headers for the specified container or blob. These operations return headers only; they do not return a response body. The URI syntax for retrieving metadata headers on a container is as follows:

**GET/HEAD https://myaccount.blob.core.windows.net/mycontainer?restype=container**

**GET/HEAD https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=metadata**

### Setting Metadata Headers

The PUT operation sets metadata headers on the specified container or blob, overwriting any existing metadata on the resource. Calling PUT without any headers on the request clears all existing metadata on the resource.

The URI syntax for setting metadata headers on a container is as follows:

**PUT https://myaccount.blob.core.windows.net/mycontainer?comp=metadata&restype=container**

**PUT https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=metadata**

### Standard HTTP properties for containers and blobs

Containers and blobs also support certain standard HTTP properties. Properties and metadata are both represented as standard HTTP headers; the difference between them is in the naming of the headers. Metadata headers are named with the header prefix x-ms-meta- and a custom name. Property headers use standard HTTP header names, as specified in the Header Field Definitions section 14 of the HTTP/1.1 protocol specification.

The standard HTTP headers supported on containers include:

* **ETag**
* **Last-Modified**

The standard HTTP headers supported on blobs include:

* **ETag**
* **Last-Modified**
* **Content-Length**
* **Content-Type**
* **Content-MD5**
* **Content-Encoding**
* **Content-Language**
* **Cache-Control**
* **Origin**
* **Range**