Contents

[Azure Blob storage 1](#_Toc50074345)

[Storage accounts 1](#_Toc50074346)

[Types of storage accounts 1](#_Toc50074347)

[Containers 2](#_Toc50074348)

[Root Containers: 2](#_Toc50074349)

[Blobs 2](#_Toc50074350)

[Blob Snapshots 3](#_Toc50074351)

[Azure Storage Capabilities: 4](#_Toc50074352)

[Access tiers for block blob data 5](#_Toc50074353)

[Azure Storage redundancy 5](#_Toc50074354)

[Redundancy in the primary region 6](#_Toc50074355)

[Redundancy in a secondary region 6](#_Toc50074356)

[Data integrity 7](#_Toc50074357)

[Pricing and billing 7](#_Toc50074358)

[Control access to account data 8](#_Toc50074359)

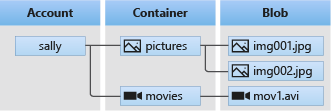
# Azure Blob storage

<https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction>

Blob storage offers three types of resources:

* The **storage account**
* A **container** in the storage account
* A **blob** in a container

The following diagram shows the relationship between these resources.



## 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. The combination of the account name and the Azure Storage blob endpoint forms the base address for the objects in your storage account.

For example, if your storage account is named *mystorageaccount*, then the default endpoint for Blob storage is:

http://mystorageaccount.blob.core.windows.net

### Types of storage accounts

<https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview>

Azure Storage offers several types of storage accounts. Each type supports different features and has its own pricing model. Consider these differences before you create a storage account to determine the type of account that is best for your applications. The types of storage accounts are:

* **General-purpose v2 accounts**: Basic storage account type for blobs, files, queues, and tables. Recommended for most scenarios using Azure Storage.
* **General-purpose v1 accounts**: Legacy account type for blobs, files, queues, and tables. Use general-purpose v2 accounts instead when possible.
* **BlockBlobStorage accounts**: Storage accounts with premium performance characteristics for block blobs and append blobs. Recommended for scenarios with high transactions rates, or scenarios that use smaller objects or require consistently low storage latency.
* **FileStorage accounts**: Files-only storage accounts with premium performance characteristics. Recommended for enterprise or high performance scale applications.
* **BlobStorage accounts**: Legacy Blob-only storage accounts. Use general-purpose v2 accounts instead when possible.

## Containers

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.

A container name must be a valid DNS name, conforming to the following naming rules:

* Container names must start or end with a letter or number, and can contain only letters, numbers, and the dash (-) character.
* Every dash (-) character must be immediately preceded and followed by a letter or number; consecutive dashes are not permitted in container names.
* All letters in a container name must be lowercase.
* Container names must be from 3 through 63 characters long.

For example, the following list shows valid and unique blob names.

* /a
* /a.txt
* /a/b
* /a/b.txt

### Root Containers:

<https://docs.microsoft.com/en-us/rest/api/storageservices/working-with-the-root-container>

## Blobs

Once the blob has been created, its type cannot be changed, and it can be updated only by using operations appropriate for that blob type, i.e., writing a block or list of blocks to a block blob, appending blocks to a append blob, and writing pages to a page blob.

All blobs reflect committed changes immediately. Each version of the blob has a unique tag, called an ETag, that you can use with access conditions to assure you only change a specific instance of the blob.

Any blob can be leased for exclusive write access. When a blob is leased, only calls that include the current lease ID can modify the blob or (for block blobs) its blocks.

A blob name must conforming to the following naming rules:

 Blob names are case-sensitive.

 Reserved URL characters must be properly escaped.

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 store up to about 4.75 TiB of data. Larger block blobs are available in preview, up to about 190.7 TiB

New blocks remain in an uncommitted state until they are specifically committed or discarded. You can upload blocks in any order, and determine their sequence in the final block list commitment step. You can also upload a new block to replace an existing uncommitted block of the same block ID. You have one week to commit blocks to a blob before they are discarded. All uncommitted blocks are also discarded when a block list commitment operation occurs but does not include them.

Block ID values can be duplicated in different blobs.

* **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.

Updating or deleting of existing blocks is not supported. Unlike a block blob, an append blob does not expose its block IDs.

* **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. For more information about page blobs, see [Overview of Azure page blobs](https://docs.microsoft.com/en-gb/azure/storage/blobs/storage-blob-pageblob-overview)

<https://docs.microsoft.com/en-us/rest/api/storageservices/understanding-block-blobs--append-blobs--and-page-blobs>

### Blob Snapshots

A snapshot is a read-only version of a blob stored as it was at the time the snapshot was created. You can use snapshots to create a backup or checkpoint of a blob. A snapshot blob name includes the base blob URI plus a date-time value that indicates when the snapshot was created.

For example, assume that a blob has the following URI:

https://myaccount.blob.core.windows.net/mycontainer/myblob

The URI for a snapshot of that blob is formed as follows:

[https://myaccount.blob.core.windows.net/mycontainer/myblob?snapshot=<DateTime](https://myaccount.blob.core.windows.net/mycontainer/myblob?snapshot=%3cDateTime)>

# Azure Storage Capabilities:

<https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview>

| **Storage account type** | **Supported services** | **Supported performance tiers** | **Supported access tiers** | **Replication options** | **Deployment model**  **1** | **Encryption**  **2** |
| --- | --- | --- | --- | --- | --- | --- |
| General-purpose V2 | Blob, File, Queue, Table, Disk, and Data Lake Gen2  6 | Standard, Premium  5 | Hot, Cool, Archive  3 | LRS, GRS, RA-GRS, ZRS, GZRS (preview), RA-GZRS (preview)  4 | Resource Manager | Encrypted |
| General-purpose V1 | Blob, File, Queue, Table, and Disk | Standard, Premium  5 | N/A | LRS, GRS, RA-GRS | Resource Manager, Classic | Encrypted |
| BlockBlobStorage | Blob (block blobs and append blobs only) | Premium | N/A | LRS, ZRS  4 | Resource Manager | Encrypted |
| FileStorage | File only | Premium | N/A | LRS, ZRS  4 | Resource Manager | Encrypted |
| BlobStorage | Blob (block blobs and append blobs only) | Standard | Hot, Cool, Archive  3 | LRS, GRS, RA-GRS | Resource Manager | Encrypted |

## Access tiers for block blob data

<https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blob-storage-tiers>

The available access tiers are:

* The **Hot** access tier. This tier is optimized for frequent access of objects in the storage account. Accessing data in the hot tier is most cost-effective, while storage costs are higher. New storage accounts are created in the hot tier by default.
* The **Cool** access tier. This tier is optimized for storing large amounts of data that is infrequently accessed and stored for at least 30 days. Storing data in the cool tier is more cost-effective, but accessing that data may be more expensive than accessing data in the hot tier.
* The **Archive** tier. This tier is available only for individual block blobs. The archive tier is optimized for data that can tolerate several hours of retrieval latency and that will remain in the archive tier for at least 180 days. The archive tier is the most cost-effective option for storing data. However, accessing that data is more expensive than accessing data in the hot or cool tiers.

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

|  | **Premium performance** | **Hot tier** | **Cool tier** | **Archive tier** |
| --- | --- | --- | --- | --- |
| **Availability** | 99.9% | 99.9% | 99% | Offline |
| **Availability**  **(RA-GRS reads)** | N/A | 99.99% | 99.9% | Offline |
| **Usage charges** | Higher storage costs, lower access, and transaction cost | Higher storage costs, lower access, and transaction costs | Lower storage costs, higher access, and transaction costs | Lowest storage costs, highest access, and transaction costs |
| **Minimum object size** | N/A | N/A | N/A | N/A |
| **Minimum storage duration** | N/A | N/A | 30 days1 | 180 days |
| **Latency**  **(Time to first byte)** | Single-digit milliseconds | milliseconds | milliseconds | hours2 |

## Azure Storage redundancy

<https://docs.microsoft.com/en-us/azure/storage/common/storage-redundancy>

The factors that help determine which redundancy option you should choose include:

* How your data is replicated in the **primary region**
* Whether your data is replicated to a **second region that is geographically distant** to the primary region, to protect against regional disasters
* Whether your application requires **read access to the replicated data** in the secondary region if the primary region becomes unavailable for any reason

### Redundancy 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. LRS is a good choice for the following scenarios:

* If your application stores data that can be easily reconstructed if data loss occurs, you may opt for LRS.
* If your application is restricted to replicating data only within a country or region due to data governance requirements, you may opt for LRS.

 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

When designing applications for ZRS, follow practices for transient fault handling, including implementing **retry policies with exponential back-off**.

We also recommend using ZRS if you want to restrict an application to replicate data only within a country or region because of data governance requirements.

### Redundancy in 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. When data is written to the secondary location, it's also replicated within that location 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. When data is written to the secondary location, it's also replicated within that location using LRS.

NOTE : The advantage of RA-GRS or RA-GZRS is that data can be read from replicas without Microsoft or the user initiating the failover.

| **Outage scenario** | **LRS** | **ZRS** | **GRS/RA-GRS** | **GZRS/RA-GZRS** |
| --- | --- | --- | --- | --- |
| A node within a data center becomes unavailable | Yes | Yes | Yes | Yes |
| An entire data center (zonal or non-zonal) becomes unavailable | No | Yes | Yes1 | Yes |
| A region-wide outage occurs in the primary region | No | No | Yes1 | Yes1 |
| Read access to the secondary region is available if the primary region becomes unavailable | No | No | Yes (with RA-GRS) | Yes (with RA-GZRS) |

1 Account failover is required to restore write availability if the primary region becomes unavailable. For more information, see [Disaster recovery and storage account failover](https://docs.microsoft.com/en-us/azure/storage/common/storage-disaster-recovery-guidance).

NOTE:

1. The primary **difference between GRS and GZRS** is how data is replicated in the primary region. Within the secondary region, data is always replicated synchronously three times using LRS. LRS in the secondary region protects your data against hardware failures.
2. For read access to the secondary region, configure your storage account to use read-access geo-redundant storage (RA-GRS) or read-access geo-zone-redundant storage (RA-GZRS). For more information, see [Read access to data in the secondary region](https://docs.microsoft.com/en-us/azure/storage/common/storage-redundancy#read-access-to-data-in-the-secondary-region).
3. Only general-purpose v2 storage accounts support GZRS and RA-GZRS.
4. To determine which write operations have been replicated to the secondary region, your application can check the **Last Sync Time** property for your storage account. All write operations written to the primary region prior to the last sync time have been successfully replicated to the secondary region, meaning that they are available to be read from the secondary.

## Data integrity

Azure Storage regularly verifies the integrity of data stored using cyclic redundancy checks (CRCs). If data corruption is detected, it is repaired using redundant data. Azure Storage also calculates checksums on all network traffic to detect corruption of data packets when storing or retrieving data.

## Pricing and billing

All storage accounts use a pricing model for Block blob storage based on the tier of each blob. Keep in mind the following billing considerations:

* **Storage costs**: In addition to the amount of data stored, the cost of storing data varies depending on the access tier. The per-gigabyte cost decreases as the tier gets cooler.
* **Data access costs**: Data access charges increase as the tier gets cooler. For data in the cool and archive access tier, you're charged a per-gigabyte data access charge for reads.
* **Transaction costs**: There's a per-transaction charge for all tiers that increases as the tier gets cooler.
* **Geo-Replication data transfer costs**: This charge only applies to accounts with geo-replication configured, including GRS and RA-GRS. Geo-replication data transfer incurs a per-gigabyte charge.
* **Outbound data transfer costs**: Outbound data transfers (data that is transferred out of an Azure region) incur billing for bandwidth usage on a per-gigabyte basis, consistent with general-purpose storage accounts.
* **Changing the access tier**: Changing the account access tier will result in tier change charges for access tier inferred blobs stored in the account that don't have an explicit tier set. For information on changing the access tier for a single blob, refer to [Blob-level tiering billing](https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blob-storage-tiers?tabs=azure-portal#blob-level-tiering-billing).

## Control access to account data

You can grant access to the data in your storage account using any of the following approaches:

* **Azure Active Directory:** Use Azure Active Directory (Azure AD) credentials to authenticate a user, group, or other identity for access to blob and queue data. If authentication of an identity is successful, then Azure AD returns a token to use in authorizing the request to Azure Blob storage or Queue storage. For more information, see [Authenticate access to Azure Storage using Azure Active Directory](https://docs.microsoft.com/en-us/azure/storage/common/storage-auth-aad).
* **Shared Key authorization:** Use your storage account access key to construct a connection string that your application uses at runtime to access Azure Storage. The values in the connection string are used to construct the Authorization header that is passed to Azure Storage. For more information, see [Configure Azure Storage connection strings](https://docs.microsoft.com/en-us/azure/storage/common/storage-configure-connection-string).
* **Shared access signature:** Use a shared access signature to delegate access to resources in your storage account, if you aren't using Azure AD authorization. A shared access signature is a token that encapsulates all of the information needed to authorize a request to Azure Storage on the URL. You can specify the storage resource, the permissions granted, and the interval over which the permissions are valid as part of the shared access signature. For more information, see [Using shared access signatures (SAS)](https://docs.microsoft.com/en-us/azure/storage/common/storage-sas-overview).

FAQ:

1. **Should I use Blob storage or GPv2 accounts if I want to tier my data?**

We recommend you use GPv2 instead of Blob storage accounts for tiering. GPv2 support all the features that Blob storage accounts support plus a lot more. Pricing between Blob storage and GPv2 is almost identical, but some new features and price cuts will only be available on GPv2 accounts. GPv1 accounts don't support tiering.

You can easily convert an existing Blob storage or GPv1 account to GPv2 through a simple one-click process.

1. **Can I store objects in all three (hot, cool, and archive) access tiers in the same account?**

Yes. The **Access Tier** attribute set at the account level is the default account tier that applies to all objects in that account without an explicit set tier. Blob-level tiering allows you to set the access tier on at the object level regardless of what the access tier setting on the account is. Blobs in any of the three access tiers (hot, cool, or archive) may exist within the same account.

1. **Can I change the default access tier of my Blob or GPv2 storage account?**

Yes, you can change the default account tier by setting the **Access tier** attribute on the storage account. Changing the account tier applies to all objects stored in the account that don't have an explicit tier set (for example, **Hot (inferred)** or **Cool (inferred)**). Toggling the account tier from hot to cool incurs write operations (per 10,000) for all blobs without a set tier in GPv2 accounts only and toggling from cool to hot incurs both **read operations** (per 10,000) and **data retrieval** (per GB) charges for all blobs in Blob storage and GPv2 accounts.

1. **Can I set my default account access tier to archive?**

No. Only hot and cool access tiers may be set as the default account access tier. Archive can only be set at the object level. On blob upload, You specify the access tier of your choice to be hot, cool, or archive regardless of the default account tier. This functionality allows you to write data directly into the archive tier to realize cost-savings from the moment you create data in blob storage.

1. **In which regions are the hot, cool, and archive access tiers available in?**

The hot and cool access tiers along with blob-level tiering are available in all regions. Archive storage will initially only be available in select regions. For a complete list, see [Azure products available by region](https://azure.microsoft.com/regions/services/).

1. **Are the operations among the hot, cool, and archive tiers the same?**

All operations between hot and cool are 100% consistent. All valid archive operations including GetBlobProperties, GetBlobMetadata, SetBlobTags, GetBlobTags, FindBlobsByTags, ListBlobs, SetBlobTier, and DeleteBlob are 100% consistent with hot and cool. **Blob data can't be read or modified while in the archive tier until rehydrated; only blob metadata read operations are supported while in archive.** However blob index tags can be read, set, or modified while in archive.

1. **When rehydrating a blob from archive tier to the hot or cool tier, how will I know when rehydration is complete?**

During rehydration, you may use the get blob properties operation to poll the **Archive Status** attribute and confirm when the tier change is complete. The status reads "rehydrate-pending-to-hot" or "rehydrate-pending-to-cool" depending on the destination tier. Upon completion, the archive status property is removed, and the **Access Tier** blob property reflects the new hot or cool tier. See [Rehydrate blob data from the archive tier](https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blob-rehydration) to learn more.

1. **After setting the tier of a blob, when will I start getting billed at the appropriate rate?**

Each blob is always billed according to the tier indicated by the blob's **Access Tier** property. When you set a new online tier for a blob, the **Access Tier** property immediately reflects the new tier for all transitions. However, rehydrating a blob from the offline archive tier to a hot or cool tier can take several hours. In this case, you're billed at archive rates until rehydration is complete, at which point the **Access Tier** property reflects the new tier. Once rehydrated to the online tier, you're billed for that blob at the hot or cool rate.

1. **How do I determine if I'll incur an early deletion charge when deleting or moving a blob out of the cool or archive tier?**

Any blob that is deleted or moved out of the cool (GPv2 accounts only) or archive tier before 30 days and 180 days respectively will incur a prorated early deletion charge. You can determine how long a blob has been in the cool or archive tier by checking the **Access Tier Change Time** blob property, which provides a stamp of the last tier change. If the blob's tier was never changed, you can check the **Last Modified** blob property. For more information, see [Cool and archive early deletion](https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blob-storage-tiers?tabs=azure-portal#cool-and-archive-early-deletion).

1. **How much data can I store in the hot, cool, and archive tiers?**

Data storage along with other 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. For more information, see [Scalability and performance targets for standard storage accounts](https://docs.microsoft.com/en-us/azure/storage/common/scalability-targets-standard-account?toc=/azure/storage/blobs/toc.json).

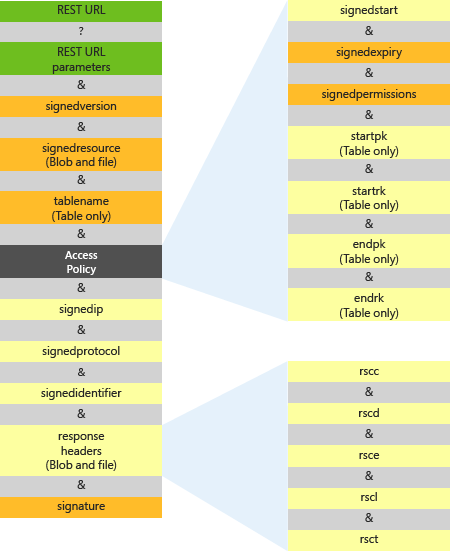
# SAS (shared access signature)

<https://docs.microsoft.com/en-us/azure/storage/common/storage-sas-overview>

A shared access signature is a signed URI that points to one or more storage resources and includes a token that contains a special set of query parameters. The token indicates how the resources may be accessed by the client.

Understanding SAS URL:

<https://docs.microsoft.com/en-us/rest/api/storageservices/create-service-sas>



GET https://myaccount.file.core.windows.net/pictures/profile.jpg?sv=2015-02-21**&**st=2015-07-01T08:49Z**&**se=2015-07-02T08:49Z**&**sr=c**&**sp=r**&**rscd=file;%20attachment**&**rsct=binary**&**sig=YWJjZGVmZw%3d%3d**&**sig=a39%2BYozJhGp6miujGymjRpN8tsrQfLo9Z3i8IRyIpnQ%3d

Resource URL : <https://myaccount.file.core.windows.net/pictures/profile.jpg>?

**Storage Services Version**: sv=2015-02-21

**SAS Validity Start Day**: st=2015-07-01T08:49Z

**SAS Validity End Day**: se=2015-07-02T08:49Z

**Allowed services**:

**Allowed resources type**: sr=c

**Allowed permission**: sp=r

**Allowed protocol**:

**Signature/signing key**: sig=YWJjZGVmZw%3d%3d

## Types of shared access signatures

Azure Storage supports three types of shared access signatures:

 **User delegation SAS.** A user delegation SAS is **secured with Azure Active Directory (Azure AD) credentials** and also by the permissions specified for the SAS. **A user delegation SAS applies to Blob storage only.**

For more information about the user delegation SAS, see [Create a user delegation SAS (REST API)](https://docs.microsoft.com/en-us/rest/api/storageservices/create-user-delegation-sas).

 **Service SAS.** A service SAS is secured with the **storage account key.** A service SAS delegates access to a resource in **only one of the Azure Storage services**: Blob storage, Queue storage, Table storage, or Azure Files.

For more information about the service SAS, see [Create a service SAS (REST API)](https://docs.microsoft.com/en-us/rest/api/storageservices/create-service-sas).

 **Account SAS.** An account SAS is secured with the **storage account key**. An account SAS delegates **access to resources in one or more of the storage services**. **All of the operations available via a service or user delegation SAS are also available via an account SAS.** Additionally, with the account SAS, you can delegate access to operations that apply at the level of the service, such as **Get/Set Service Properties** and **Get Service Stats** operations. You can also delegate access to read, write, and delete operations on blob containers, tables, queues, and file shares that are not permitted with a service SAS.

For more information about the account SAS, [Create an account SAS (REST API)](https://docs.microsoft.com/en-us/rest/api/storageservices/create-account-sas).

| **Type of SAS** | **Type of authorization** |
| --- | --- |
| User delegation SAS (Blob storage only) | Azure AD |
| Service SAS | Shared Key |
| Account SAS | Shared Key |

## Best practices when using SAS

The following recommendations for using shared access signatures can help mitigate these risks:

* **Always use HTTPS** to create or distribute a SAS. If a SAS is passed over HTTP and intercepted, an attacker performing a man-in-the-middle attack is able to read the SAS and then use it just as the intended user could have, potentially compromising sensitive data or allowing for data corruption by the malicious user.
* **Use a user delegation SAS when possible.** A user delegation SAS provides superior security to a service SAS or an account SAS. A user delegation SAS is secured with Azure AD credentials, so that you do not need to store your account key with your code.
* **Have a revocation plan in place for a SAS.** Make sure you are prepared to respond if a SAS is compromised.
* **Define a stored access policy for a service SAS.** Stored access policies give you the option to revoke permissions for a service SAS without having to regenerate the storage account keys. Set the expiration on these very far in the future (or infinite) and make sure it's regularly updated to move it farther into the future.
* **Use near-term expiration times on an ad hoc SAS service SAS or account SAS.** In this way, even if a SAS is compromised, it's valid only for a short time. This practice is especially important if you cannot reference a stored access policy. Near-term expiration times also limit the amount of data that can be written to a blob by limiting the time available to upload to it.
* **Have clients automatically renew the SAS if necessary.** Clients should renew the SAS well before the expiration, in order to allow time for retries if the service providing the SAS is unavailable. If your SAS is meant to be used for a small number of immediate, short-lived operations that are expected to be completed within the expiration period, then this may be unnecessary as the SAS is not expected to be renewed. However, if you have client that is routinely making requests via SAS, then the possibility of expiration comes into play. The key consideration is to balance the need for the SAS to be short-lived (as previously stated) with the need to ensure that the client is requesting renewal early enough (to avoid disruption due to the SAS expiring prior to successful renewal).
* **Be careful with SAS start time.** If you set the start time for a SAS to the current time, you may observe failures occurring intermittently for the first few minutes due to different machines having slight variations the current time (known as clock skew). In general, set the start time to be at least 15 minutes in the past. Or, don't set it at all, which will make it valid immediately in all cases. The same generally applies to expiry time as well--remember that you may observe up to 15 minutes of clock skew in either direction on any request. For clients using a REST version prior to 2012-02-12, the maximum duration for a SAS that does not reference a stored access policy is 1 hour, and any policies specifying longer term than that will fail.
* **Be careful with SAS datetime format.** If you set the start time and/or expiry for a SAS, for some utilities (for example for the command-line utility AzCopy) you need the datetime format to be '+%Y-%m-%dT%H:%M:%SZ', specifically including the seconds in order for it to work using the SAS token.
* **Be specific with the resource to be accessed.** A security best practice is to provide a user with the minimum required privileges. If a user only needs read access to a single entity, then grant them read access to that single entity, and not read/write/delete access to all entities. This also helps lessen the damage if a SAS is compromised because the SAS has less power in the hands of an attacker.
* **Understand that your account will be billed for any usage, including via a SAS.** If you provide write access to a blob, a user may choose to upload a 200 GB blob. If you've given them read access as well, they may choose to download it 10 times, incurring 2 TB in egress costs for you. Again, provide limited permissions to help mitigate the potential actions of malicious users. Use short-lived SAS to reduce this threat (but be mindful of clock skew on the end time).
* **Validate data written using a SAS.** When a client application writes data to your storage account, keep in mind that there can be problems with that data. If your application requires that data be validated or authorized before it is ready to use, you should perform this validation after the data is written and before it is used by your application. This practice also protects against corrupt or malicious data being written to your account, either by a user who properly acquired the SAS, or by a user exploiting a leaked SAS.
* **Know when not to use a SAS.** Sometimes the risks associated with a particular operation against your storage account outweigh the benefits of using a SAS. For such operations, create a middle-tier service that writes to your storage account after performing business rule validation, authentication, and auditing. Also, sometimes it's simpler to manage access in other ways. For example, if you want to make all blobs in a container publicly readable, you can make the container Public, rather than providing a SAS to every client for access.
* **Use Azure Monitor and Azure Storage logs to monitor your application.** You can use Azure Monitor and storage analytics logging to observe any spike in authorization failures due to an outage in your SAS provider service or to the inadvertent removal of a stored access policy. For more information, see [Azure Storage metrics in Azure Monitor](https://docs.microsoft.com/en-us/azure/storage/common/monitor-storage?toc=/azure/storage/blobs/toc.json) and [Azure Storage Analytics logging](https://docs.microsoft.com/en-us/azure/storage/common/storage-analytics-logging?toc=/azure/storage/blobs/toc.json).